ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

INGENIERIA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Sistema de Reconocimiento y Clasificación de Objetos Usando Cámaras RGB-D Para Manipulación Robótica

TRABAJO FIN DE GRADO

Leganés, Junio 2014

AUTOR: MIGUEL ÁNGEL SANZ SARDINA

DIRECTOR: SILVIA RODRÍGUEZ JIMÉNEZ

TUTOR: MOHAMED ABDERRAHIM FICHOUCHE

AGRADECIMIENTOS

En primer lugar quiero agradecer al profesor Abderrahim su amabilidad para facilitarme cual tipo de ayuda de su personal así como del material del laboratorio que tan generosamente me ha prestado durante la duración del todo el proyecto. Gracias también por dedicar tanto esfuerzo y esmero en seguir y colaborar en la corrección de la memoria. Muchísimas gracias Mohamed.

Si tengo que decir una persona que de verdad haya permitido llevar a cabo este proyecto, esa sería mi directora de proyecto Silvia Rodríguez. Desde que hace un año la pretendí de palabra para tutelar mi TFG hasta que exponga mi proyecto he sentido su cercanía y empatía, he recibido siempre su ayuda y apoyo, y he notado toda su confianza. La admiro como profesora, como profesional y como persona. Gracias por todo Silvia.

Gracias a mi colegio por darme los mejores años de mi infancia. A la directora Madre Ángeles y todo el claustro que siempre me han tratado tan bien. Os admiro a todos.

Quiero agradecer a todos mis cercanos que durante toda la carrera me han dado aliento, fuerzas, empujoncitos de ánimo, risas y buenos momentos y muchísimo apoyo. A todos: a Jose (buen ingeniero y mejor persona), a mis amigos y profes de tenis (Victor Palacios, Patri y David), a todos mis amigos increíbles de la “Chupipandi” (Ana, Apo, Francis, Macarena, Maka, Nuri, Luis y Raquel) y a un amigo súper especial como Juanito y toda su maravillosa familia.

El último lugar es para los más importantes, para los que no se han perdido ni un éxito ni un fracaso en mi vida. Aquellos que sólo me han dado amor, amor y más amor. Gracias por todo lo que aquí no escribo pero que sí sabéis. Gracias a mis padre y a mis 6 hermanos: GRACIAS Mama, GRACIAS Papá, GRACIAS Sarita, GRACIAS Gabi, GRACIAS Luisito, GRACIAS Alfonsito, GRACIAS Nico y GRACIAS Kaki. Os quiero <3

RESUMEN

La manipulación de objetos mediante una mano robótica requiere dotar a la plataforma de una capacidad de percepción avanzada. El presente Trabajo Fin de Grado (TFG) aborda esta temática desde el punto de vista de reconocimiento de objetos en entornos domésticos mediante el uso de una cámara Kinect, que proporciona datos tanto de color (RGB) como de profundidad (Depth).

Este documento recopila los diferentes métodos de visión existentes para el reconocimiento de objetos en el contexto abordado, explicando paso a paso las diferentes etapas. Los métodos evalúan tanto la información de color como de profundidad, buscando patrones cualitativos para el reconocimiento de los objetos de interés. Además, esta memoria recoge los resultados experimentales de las primeras etapas del reconocimiento de objetos desarrollado, empleando una sola imagen RGB-D de la Kinect: visualización y segmentación de los objetos de interés colocados sobre una mesa. Para ello, se emplea el entorno Robot Operating System (ROS).

Por último se explicarán las conclusiones alcanzadas tras el estudio teórico y los resultados experimentales, incluyendo una serie de propuestas y mejoras a realizar como trabajo futuro.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Contexto

El reconocimiento de objetos es uno de los principales desafíos a los que se está enfrentando la actual revolución tecnológica en la que se encuentra nuestra sociedad. Y es que es un hecho que el mundo se está automatizando a un ritmo vertiginoso. Dentro de ese proceso tan complejo el reconocimiento de los objetos juega un papel fundamental, siendo una de las etapas que más restricciones o libertades puede otorgar.

Es por eso que actualmente este campo está en pleno crecimiento, porque las herramientas que permiten la observación del entorno está sufriendo un desarrollo fortísimo, no limitándose a sensores o detectores físicos, sino también a enormes adquisiciones de datos de cualquier tipo y recreaciones computacionales, aplicación e incorporación de complejos algoritmos y complejos métodos de identificación,…

En este proyecto se abordará el reconocimiento básico de objetos aplicado a robótica para manipulación con un brazo robot mediante el uso de una herramienta (una cámara Kinect) muy popular en el mundo de las videoconsolas (uno de los sectores que más fuerte está apostando por las ventajas y novedades que puede aportarles el reconocimiento de objetos a su producto final).

Es por eso que el entorno que usaremos será aquel que esté totalmente enfocado y desarrollado para nuestro objetivo, siendo Robot Operating System[1] (ROS) versión Fuerte el middleware elegido por motivos que más adelante se expondrán.

Además es importante destacar que gran parte del software usado es de carácter “Opensource” [1] (código abierto [2]) permitiéndonos incorporar contribuciones de cualquier persona que haya trabajado este campo.

[1] http//www.ros.org/

1.1.1 Proyecto HANDLE

El proyecto HANDLE se trata de una colaboración a nivel europeo entre diferentes instituciones de investigación el cual tenía por objetivo el desarrollo completo de un robot que tuviera la capacidad de aprender e imitar a humanos y realizar tareas similares con la autonomía deseada.

Los cinco objetivos que centraba HANDLE eran:

- Caracterización de “object affordances”.
- Aprendizaje de tareas y actividades directamente a través de la imitación de seres humanos.
- Optimización y desarrollo de las aptitudes del robot a través de la interacción.
- Manipulación diestra autónoma.
- Desarrollo de manos artificiales similares a dispositivos de fácil instalación y uso (plug-in).

Para llevar a cabo estos cinco objetivos se subdividió el proyecto en paquetes denominados “WorkPackage”, los cuales se enumeran a continuación:

- Grabación y codificación.
- Aprendizaje de estrategias humanas de manipulación.
- Mejora de habilidades.
- Percepción multimodal.
- Arquitectura de control.
- Hardware del robot.
- Tecnologías para manos robóticas futuras.
- Evaluación y difusión de resultados y formación.
- Coordinación.

Este TFG aborda, de una forma básica, el apartado de percepción multimodal, ya que la temática expuesta trata la clasificación y la percepción de objetos.

1.2 Objetivos

El objetivo principal de este proyecto es, en primera instancia, el reconocimiento 3D de objetos a través de una cámara Kinect sacando todo el provecho de la tecnología que esta herramienta incorpora. En una segunda instancia es conseguir incorporar bases de datos externas a través de la cuales la identificación de objetos sea más amplia, completa, efectiva y exitosa.

Los objetivos del proyecto, enumerados a continuación, son muy sencillos pero a su vez instructivos y cargados de nuevos conocimientos que sirven de introducción al actual y avanzado mundo del reconocimiento y percepción del entorno:

1. Adquirir soltura en el manejo de las principales funciones en el entorno de ROS, familiarizándose con las herramientas y funciones que por defecto incluye y que están ligadas al reconocimiento de objetos.
2. Ser capaces de incorporar stacks[2] desarrollados por otras personas con el fin de crear un conjunto que nos permita llevar a cabo una función en concreto cuya finalidad será un subproceso o etapa dentro del reconocimiento de objetos.
3. Aplicación de las principales técnicas de reconocimiento que podemos hacer uso mediante una cámara Kinect.
4. Identificación de objetos de geometría sencilla que estarán ubicados sobre una superficie plana a una distancia reducida.
5. Incorporación de una base de datos que permita una mayor velocidad de reconocimiento.
6. Incorporación e implementación de bases de datos de otros usuarios lo que se traducirá en mejores y más potentes resultados dentro del reconocimiento.

[2] Paquetes en español o subprogramas que se ejecutan como módulos de un programa más complejo

1.3 Factores limitativos

A continuación se expondrá los principales factores limitativos o restricciones que tendrá el proyecto y que determinarán la extensión y profundidad del mismo.

1.3.1 Factor temporal

Como en todo proyecto, se han establecido unas fechas orientativas para las cuales se debe, al menos, haber cumplido con el objetivo principal del proyecto. Esta organización temporal queda recogida en un diagrama de Grantt (Figura 1).

Dado que se trata de un TFG (Trabajo Fin de Grado) y que debe ser expuesto para valoración ante jurado en una fecha ya concretada, la limitación temporal al avance y desarrollo del mismo lo haremos en el punto de no retorno más próximo a una semana antes de la entrega obligada del TFG.

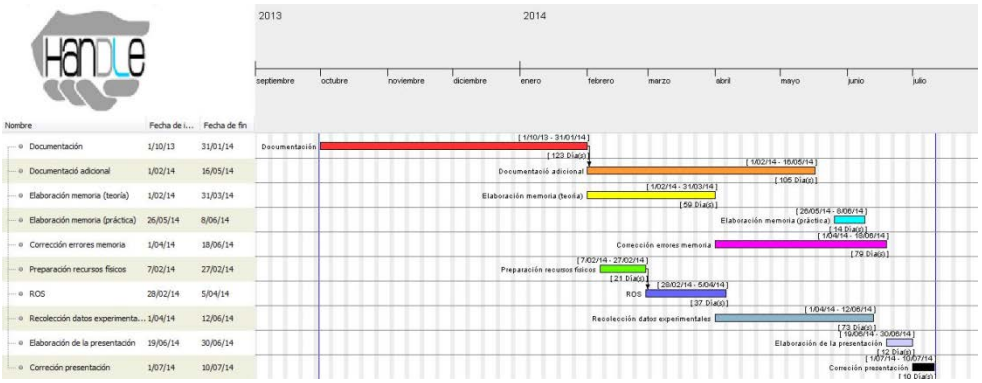

Figura 1. Potencial de todas las herramientas y cualidades de ROS

1.3.2 Factor de comprensión

El mundo de la visión por computadora está en auge y expansión, cada vez a una velocidad más vertiginosa.

Aunque este proyecto utiliza un hardware y software en concreto, las posibilidades del mismo hoy en día aún no ha tocado techo debido al constante aporte y desarrollo de la comunidad.

Por tanto, aunque se parte de trabajo previo, mis conocimientos de visión por computador al comienzo de este proyecto son ínfimos por lo que el objetivo fundamental de este proyecto es adquirir y afianzar las nociones de visión por computador necesarias para lograr un funcionamiento básico de la Kinect, centrándose en el reconocimiento 3D mediante ROS. Este estudio conlleva una búsqueda de las técnicas que existen, una amplia labor de documentación así como pruebas de ensayo-error-aprendizaje para asentar los conocimientos en un campo totalmente nuevo para mí. Por este motivo, el funcionamiento avanzado no se completa en este TFG pero se dejarán todas las bases asentadas para futuros proyectos.

1.4 Recursos hardware y software

A continuación se detallarán requisitos técnicos del hardware y software a usar, y más adelante, en otro epígrafe se dará información a nivel de producto.

1.4.1 Hardware

El uso de ROS implica unos requisitos mínimos recomendados en el PC para el correcto funcionamiento del mismo:

a) Procesador Intel/AMD de dos núcleos o más

b) Velocidad de reloj de al menos 2.2Ghz

c) RAM mínima de 2Gb

d) Espacio en el disco duro de 60Gb (incluida instalación de ROS).

e) Tarjeta Gráfica con al menos 1Gb de memoria dedicada

Además de los requisitos impuestos por ROS, necesitaremos contar con una cámara Kinect RGB-D (en su primera versión comercial para Xbox 360[3]).

1.4.2 Software

A continuación se especifican los programas, herramientas y librerías que se han usado en este TFG:

a) S.O Ubuntu versión 12.1

b) Framework ROS Fuerte

c) Librerías C, C++, Python y PCL.

d) Software de desarrollo Visual C++

[3] Noviembre de 2010

2. SOFTWARE Y HARDWARE

2.1 Robot Operating System (ROS)

El acrónimo de ROS significa “Robot Operating System”, que se traduce por Sistema Operativo para Robots. Como su nombre indica se trata de una infraestructura digital que incorpora los procesos tanto básicos como complejos que se requieren para el control y desarrollo de funciones robóticas de cualquier tipo (Figura 2).

ROS es software libre bajo términos de la licencia BSD (Berkeley Software Distribution). Esto le ha otorgado una posición privilegiada y favorecida para la rápida expansión y desarrollo por parte de la comunidad. Esto se hace posible porque ROS se fundamenta en paquetes gestionados por ros-pkg, los cuales son aportados por los usuarios y a los que todos aquellos que deseen tienen acceso y privilegios de uso, modificación y desarrollo (filosofía de “Opensource”).

Los principales servicios que nos permite ROS es el control de dispositivos a bajo nivel, comunicación entre procesos, mensajes a través de paquetes, localización y mapeos simultáneos, planificación, percepción, simulación,…

Figura 2. Potencial de todas las herramientas y cualidades de ROS

ROS cuenta con varias versiones que de forma periódica se van mejorando y desarrollando. La versión original recibió el nombre de ROS 1.0 y nació para el público en enero de 2010. Posteriormente se actualizó y surgieron otras versiones como Turtle, Diamondback, Electric, Fuerte, Galapagos y por último y más reciente Hydra. En este proyecto se usará ROS Fuerte, ya que el objetivo del proyecto es continuación del proyecto HANDLE [17] el cual usaba esta misma versión de ROS, y con la finalidad de no tener problemas de implementación y compatibilidad se ha

decidido usar el mismo software y versiones.

2.2 Cámara Microsoft Kinect

La cámara Kinect[4] (Figura 3a y Figura 3b) se trata de un controlador de juego libre que se incluye como accesorio de la videoconsola Xbox 360, desarrollada por Microsoft [1]. Fue lanzada en noviembre de 2010 y posteriores versiones han sido lanzadas con mejoradas funciones.

Figura 3a. Cámara Kinect de la XBOX 360 diseñada por Microsoft

La Kinect tiene a sus espaldas 20 años de desarrollo e investigación. Durante esta fase recibía el nombre en clave “Proyecto Natal”[3]. Su propósito era revolucionar el mundo del videojuego al incluir la participación activa del jugador a través de voz y movimientos naturales.

2.2.1 Descripción física

La Kinect cuenta con un soporte móvil (Figura 4), una matriz de múltiples

[4] Microsoft: Kinect for Xbox 360. (2010)

matrices, una cámara RGB[5] y una cámara de profundidad 3D[6] (Figura 5), que a continuación se explican con más detalle:

Soporte móvil: se trata de un pie que soporta la cámara en sí y que es graduable su inclinación (articulación en rótula) con el fin de mejorar y facilitar el enfoque de los jugadores.

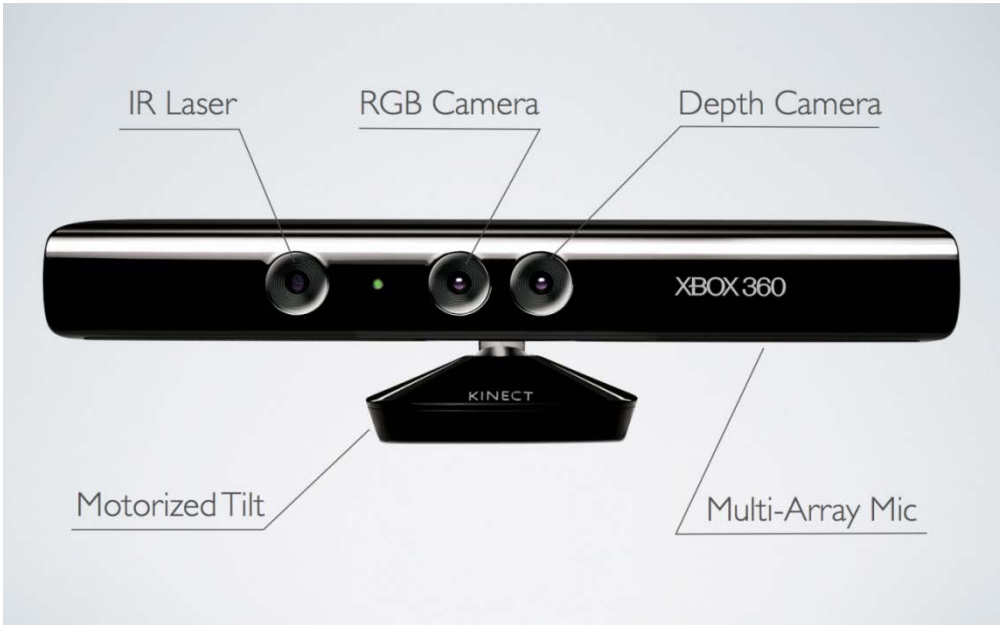

Figura 3b. Descripción de las partes visibles de la Kinect

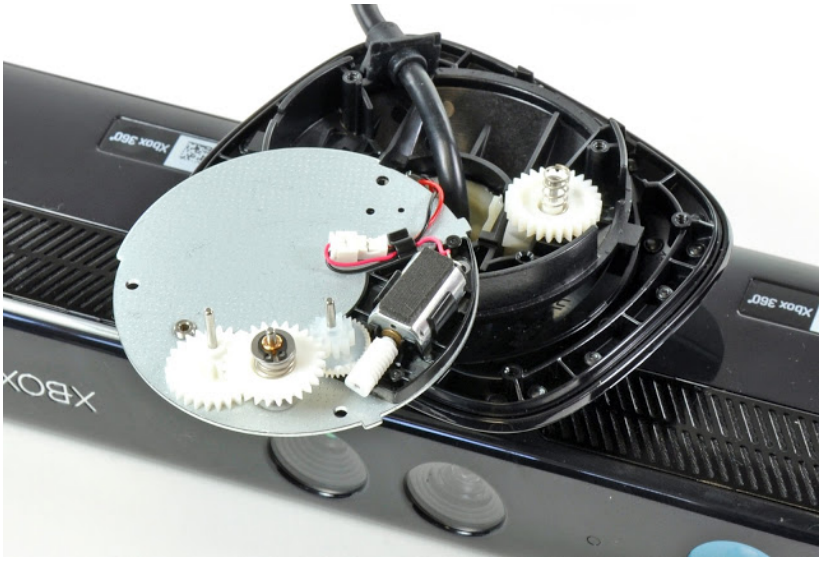

Figura 4. Mecanismo del soporte móvil de la Kinect

[5] En inglés Red, Green, Blue.
[6] 3 Dimensiones

Cámara de profundidad 3D: es una combinación de un proyector de infrarrojos y un sensor CMOS[7] monocromo que permite detectar la estancia en interiores con controlada luminosidad. El rango de detección es variable gracias a un software implementado que autocalibrar el sensor.

Cámara RGB: esta cámara simplemente provee de color a los datos obtenidos con la cámara de profundidad.

Figura 5. De izq. a der. - Sensor CMOS de imagen RGB, sensor CMOS de imágenes IR y proyector IR

2.2.2 Funcionamiento de la Kinect

La sala donde se ubica el sistema y sus ocupantes está “salpicada” con un patrón de puntos invisibles (para los usuarios) y que son generados por un láser que trabaja en la frecuencia del infrarrojo cercano. Esto es, un dispositivo láser de Clase 1 que proporciona un enfoque a una distancia útil sin generar condiciones peligrosas para los jugadores. Un sensor de imagen CMOS detecta los reflejos del haz infrarrojo, dentro de un patrón de puntos y segmentos, y genera mapas que informan sobre la intensidad de cada punto y segmento.

Este proceso se realiza a una distancia de pocos metros y se obtiene una resolución de hasta 1 centímetro dentro de la dimensión de la profundidad (Eje Z). La resolución espacial (Ejes X e Y) es del orden de milímetros, y se utiliza la entrada de información RGB (Rojo, Verde y Azul) desde un segundo sensor de imagen CMOS (como la cámara de un teléfono

[7] En inglés Complementary metal-oxide-semiconductor

móvil) para añadir color a los datos adquiridos. El frente del Kinect por lo tanto, donde se ubican los dispositivos mencionados, se ve puede apreciar en la Figura 3.

Kinect utiliza los datos obtenidos sobre la posición tridimensional y el movimiento de las personas ubicadas frente a él para reproducir y reflejar las situaciones correspondientes en la pantalla, las que son ejecutadas por el avatar de cada jugador, es decir, por el personaje de la ficción. Un conjunto de engranajes acoplados a un pequeño motor (conjunto ubicado en el pié de Kinect) mantiene al equipo y sus sensores de imágenes siempre atentos y direccionados hacia el espacio de donde proviene o se genera la acción, siguiendo y capturando la información sobre cómo se mueven los jugadores. Cuatro micrófonos se utilizan para cancelar los ecos y el ruido de fondo, mientras que ayudan a determinar cuál de todos los jugadores ha emitido un comando de voz como se ve en la Figura 6.

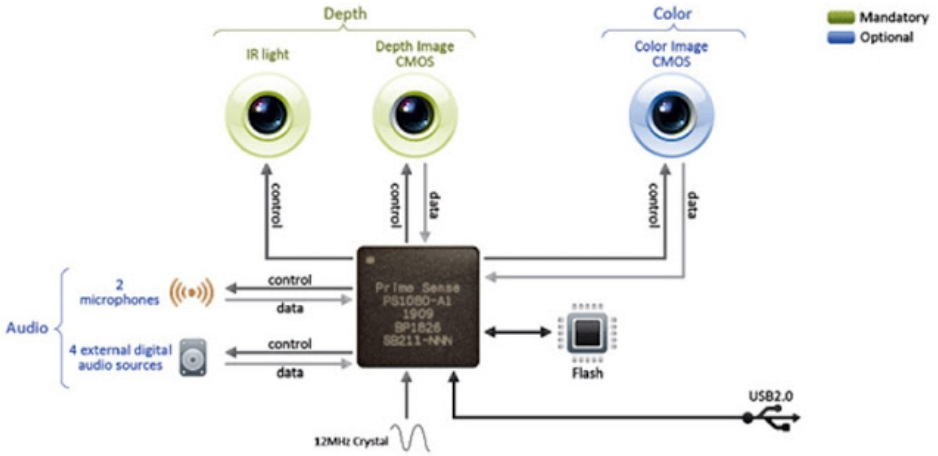

Figura 6. Esquema de comunicación y funciones de la Kinect

Los "ojos" de Kinect son un par de cámaras. Estos captores son sensores de imágenes CMOS de Aptina Imaging[8] y el modelo elegido utiliza el sensor de infrarrojos MT9M001, con píxeles de 5,2 micras. Vale aclarar que píxeles más grandes trabajan bien con poca luz y sumado esto a una correcta filtración, se prestan muy bien a las aplicaciones de infrarrojos.

Por el lado de la entrada de información RGB se utiliza una cámara color, con un sensor MT9M112 y los dos sensores mencionados poseen una

[8] Micron: Aptina Imaging. (2008).

resolución de 1,3 mega-píxeles. La interfaz utilizada para dar forma al "procesamiento sensorial" es un chip PS1080 de PrimeSense[9] (Figura 8.2 - rotulado azul), que se encarga de todos los controles del sistema, del proyector de infrarrojos, de los procesos de la información que cada cámara recoge y de las entradas de audio.

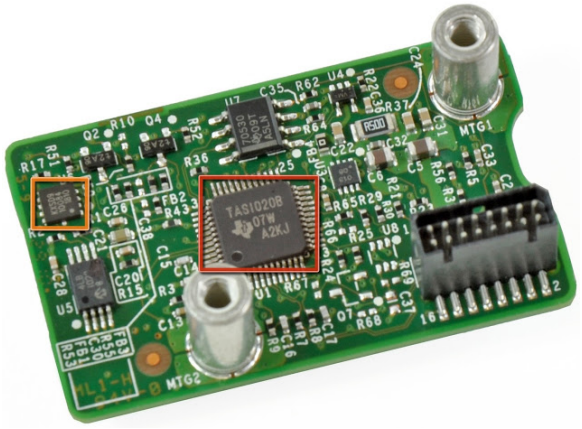

Figura 7. Placa base 1 de la Kinect

A su vez, el PS1080 establece su enlace a través de una conectividad USB 2.0 (Figura 7 – rotulado naranja) hacia uno de los procesadores principales de aplicaciones dentro de Kinect. Estamos hablando del PXA168 que puede trabajar a velocidades tan altas como los jugadores más frenéticos puedan exigirle al sistema de proceso. Completando los dispositivos de entradas, nos encontramos con un par de convertidores analógicos-digitales estéreo, WM8737L (Figura 8. – rotulado rojo) de Wolfson Microelectronics[10], encargados de realizar el trabajo de preamplificadores de audio que dan cabida a la matriz de micrófonos ya mencionada.

Figura 8. Placa base 2 de la Kinect

[9] Apple Inc.: PrimeSense (2005).
[10] Wolfson Microelectronics plc (1984)

Figura 9. Placa base 3 de la Kinect

Kinect también cuenta con un acelerómetro MEMS, el KXSD9 de Kionix[11] (Figura 7). Debido a que la unidad tiene un rango (limitado) de movimientos, este acelerómetro forma parte del bucle de control de inclinación, que también incluye el controlador para motor paso a paso, el A3906 de Allegro Microsystems. Otros componentes notables encontrados son el controlador uPD720114 que cumple la función de concentrador USB de NEC y un par de circuitos integrados de Texas Instruments: el TAS1020B (Figura 7 – rotulado rojo) que es un controlador de audio USB Streaming y un ADS7830 que es un convertidor analógico-digital de ocho canales y de 8 bits.

A continuación se va a nombrar todas las partes rotuladas de las imágenes referidas a las placas bases de la Kinect:

1. Placa base de la Kinect (Figura 7):
    - TI TAS1020B USB (rojo): Controlador de audio
    - A Kionix MEMS KXSD9 (naranja): Acelerómetro Usado posiblemente como estabilizador de imagen.
2. Placa base 2 de la Kinect (Figura 8.1):
    - Analog Devices AD8694 (rojo): Salida operacional amplificadora CMOS rail-to-rail simple,dual o cuadrática de bajo coste y bajo ruido.
    - TI ADS7830I (naranja): 8-bit, 8-channel conversor analógico-digital por muestreo con interfaz I2C.
    - Allegro Microsystems A3906 (amarillo): Controlador de motor DC simple/doble paso a paso de vajo voltaje.
    - ST Microelectronics M29W800DB (verde): 8 Mbit (1Mb x8 or 512Kb x16) Memoria flash NV.

[11] Kionix Inx by MEMS. (2008)

- PrimeSense PS1080-A2 (azul): SoC sensor procesador de imágenes.

3. Placa base 2 de la Kinect (Figura 8.2):

   - Wolfson Stereo ADC WM8737G: Conversor analógico digital de sonido estéreo de baja potencia diseñado para aplicaciones potables. Para conseguir una salida de audio de 16-32-bits utiliza ADCs sigma-delta multi-bit estéreo de 24-bits y soporta una frecuencia de muestreo de 8Hz a 96Hz.

   - Fairchild N-Channel PowerTrench MOSFET (naranja): MOSFET de N-canales diseñado especialmente para mejorar la eficiencia total de conversores DC/DC usando tanto síncronos como controladores convencionales de interconectores PWM. Se optimizo para puertas de baja carga, baja rDS y alta velocidad de conexión.

   - NEC uPD720114 USB 2.0 hub controller (amarillo).

   - H1026567 XBOX1001 X851716-005 GEPP (verde).

   - Marvell AP102 (azul): SoC con controlador para cámara Hynix H5PS5162FF 512 megabit DDR2 SDRAM

   - Hynix H5PS5162FF 512 megabit DDR2 SDRAM (rosa).

2.2.3 Datos técnicos de la Kinect

A continuación se enumeran las propiedades técnicas más relevantes de la cámara Kinect:

- Resolución:
  - Linex y Mac: 640x480, 320x240.
  - Windows 7: 1024x768, 640x480, 320x240.
- Frecuencia de hasta 30 frames/sec.

- Campo de visión:
    - Horizontal: 58º
    - Vertical: 45º
    - Diagonal: 70º
- Resolución espacial:
    - Eje X: 3mm
    - Eje Y: 3mm
    - Eje Z: 10mm
- Rango de detección y funcionamiento:
    - Xox 360: 1.2m – 3.5m
    - Windows 7: 0.4m – 3.5m
- Rango de inclinación: ±27º

2.2.4 Usos de la Kinect

La Kinect ha irrumpido en el mundo de la tecnología con más fuerza de la que se podría esperar. Es una opinión compartida por todo amante de la tecnología ya que se trata de un dispositivo con un potencial increíble para todos los campos tecnológicos con un precio de adquisición apto para todos los públicos.

A continuación se enumeran algunos de los múltiples usos [18,19] en robótica que se le da a la Kinect, además del original para el que fue concebido (mejor experiencia de juego en videoconsolas Xbox):

a) Generador de modelos 3D en tiempo real a partir del reconocimiento de objetos o extremidades. Este proyecto, desarrollado por un equipo de investigadores del Massachusetts Institute of Technology [24], recibe el nombre de inFORM [25].

b) Reconocimiento de objetos y espacios en 3D, cómo el que se aborda en este TFG.

c) Reconstructor de modelos en 3D.

d) Convertir superficies sólidas lisas (como una pared) en pantallas "touchscreen"[12].

e) Aún en fase experimental, el manejo remoto de aparatos quirúrgicos y métodos de rehabilitación a distancia [14].

[12] Táctiles, interactivas al tocarlas.

3. ESTADO DEL ARTE

Una de las necesidades de la robótica es el reconocimiento de objetos y su clasificación. Los objetivos pueden ser el trazado de un recorrido que realizará un robot o la manipulación de objetos con una mano antropomórfica.

Este proyecto está más orientado al reconocimiento para una posterior manipulación con una mano robótica.

Concretamente se ubica este TFG dentro del proyecto HANDLE cuyo uno de sus objetivos era el de reconocer objetos a cortas distancias en entornos no estructurados y así poder manipularlos con la mayor destreza posible en ausencia de práctica y experiencia.

Esta necesidad de reconocer objetos se hace evidente puesto que es imposible crear una fuente de datos con todos los objetos del entorno que puede encontrar una mano robótica. Esto sería posible si se restringe el entorno en el que se ubica el robot así como los objetos que lo componen. Pero la óptima solución (la cual caracteriza a un robot) es la autonomía e independencia sin importar en qué situación se encuentre. Y esto se alcanza desarrollando la percepción y clasificación en vez de abusar de bases de datos.

Así pues, que un robot sea capaz de percibir objetos, identificarlos, estudiarlos y clasificarlos autónomamente para su posterior manipulación con una mano robótica es lo que se desarrolla en este Trabajo Fin de Grado.

3.1 Pasos previos

La cámara Kinect provee imágenes con una cantidad de información elevadísima para su manipulación con computadores corrientes. Alrededor de 300 mil puntos son los que, si no depuramos, deberemos iterar cada vez que realicemos algún cálculo.

Es por ello que se hace necesario filtrar toda la información pasiva, invariante y no necesaria. En nuestro caso suponemos que los objetos que se va tratar de

reconocer están sobre una superficie sólida, lisa, recta y finita como podría ser una mesa.

Por tanto, será imprescindible en primera instancia detectar la mesa y en adelante ignorar la información que ella nos proporcione.

3.1.1 Detección de la superficie

Para este paso se hará uso de un algoritmo genérico de detección de superficies similares a una mesa a través de una nube de puntos tridimensionales [22,23]. Se trata del algoritmo RANSAC (Random Sample Consensus), el cual fue desarrollado por Fischler y Bolles [20] en el año 1981.

Los pasos para la detección del plano sobre el que se encuentran los objetos se enumeran a continuación:

1) Se seleccionan 3 puntos aleatoriamente de la nube de puntos. Si los 3 puntos no definen un plano inequívoco (por ejemplo son colineales) se eligen otros 3 puntos. Así hasta que se cumple este requisito.

2) Para dibujar la mesa se realizará una estimación en función de un modelo que describa superficies planas, como es el caso de la ecuación normal de Hesse: $ax + by + cz + d = 0$.

3) Se establece un umbral de distancia el cual debe ser mayor que la distancia entre los puntos que se consideren parte de la superficie plana y la misma.

4) Se repite el paso 3 tantas veces como se desee reduciendo el umbral hasta conseguir el resultado deseado. Una vez finalizado se recrea la mesa.

Todos los puntos que se han usado para recrear la mesa se excluyen de ahora en adelante (se omiten) y se usan los restantes, los cuales se supondrán que pertenecen a los objetos que estaban colocados sobre la superficie.

Cuando exista un falso positivo no debe preocupar puesto que en pasos posteriores serán automáticamente discriminados por ser incoherentes con el resto de puntos que si pertenecen a los objetos.

3.2 Método de extracción de características

Para el reconocimiento de objetos es necesario primeramente extraer las características de los objetos a evaluar. Esta es primera fase se considera de gran importancia y complejidad porque se deben seleccionar cuidadosamente los parámetros que van a definir la adquisición de datos, y de ser errónea esta elección o simplemente no lo suficiente correcta, los resultados serán negativos e incluso será imposible llegar a la fase de clasificación.

Los métodos de extracción de características pueden ser tanto en 2D como en 3D, como combinando ambos. La Kinect, al proporcionar datos tanto de color como de profundidad, permite la combinación de ambos métodos permitiendo la obtención de resultados mejores y más óptimos al explotar las características de ambos métodos.

3.2.1 Extracción de características a partir de modelos 2D

A través de algoritmos somos capaces de, con información 2D (imágenes), encontrar objetos, clasificarlos e incluso obtener sus características más generales con las cuales podremos posteriormente realizar una clasificación de los mismos.

A continuación se detallarán los algoritmos más usados (por su calidad y adaptabilidad a cualquier entorno y objeto) que proporcionan mejores resultados:

3.2.1.1 Scale-Invariant Feature Transform (SIFT)

Este algoritmo tiene por misión detectar y localizar imágenes en el ámbito local. El algoritmo definitivo fue publicado por David Lowe en 1999 [6].

El campo de aplicación de este algoritmo es bastante extenso: reconocimiento de objetos, manipulación con robots, navegación, modelado de 3D, video seguimiento,, entre otros.

El inicio de este algoritmo comprendía localizar puntos de objetos captados desde varias perspectivas (Figura 9). Luego se mejoró y se consiguió localizar vértices e identificarlos usando varias perspectivas.

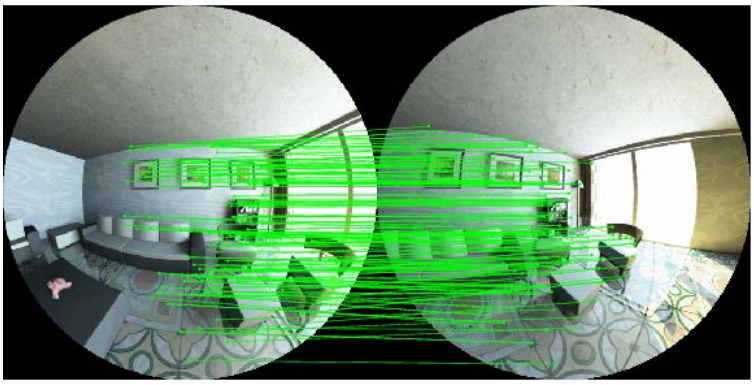

Figura 10. Identificación del mismo putos en imágenes con diferente perspectiva

El último y definitivo paso fue de manos de Lowe consiguiendo captar imágenes en las cuales era capaz de destacar los “puntos de interés”.

Los pasos de este algoritmo son:

1) Construcción de pirámides de Scale-Space: Una misma imagen se representa usando escalas y tamaños diferentes. Para que el resultado sea óptimo se usa la función de diferencia gaussiana (Figura 10), la cual es capaz de identificar los posibles puntos de interés los cuales son invariantes al tipo de representación (en este caso de tamaño, escala y orientación).

$$L(x, y, \sigma) = G(x, y, \sigma) * l(x, y) \quad (1)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^3)} \quad (2)$$

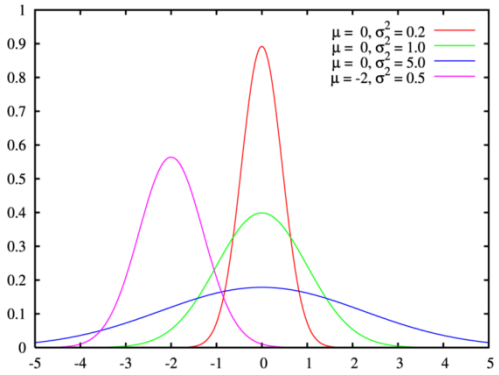

Figura 11. Curvas gausianas con diferentes parámetros

2) Localización de puntos clave: Contrastando los resultados se observa qué puntos han mantenido sus propiedades tras el cambio de escalas. Es por eso necesario analizar cada píxel y compararlo con las otras imágenes de escala diferente (Figura 11). Si son puntos estables entonces son seleccionados.

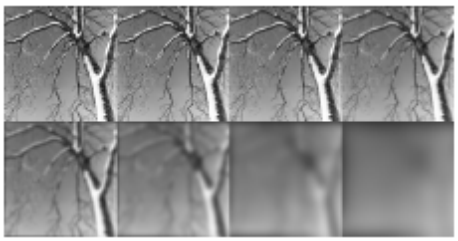

Figura 12. Imagen en diferente escalas.

3) Asignación de orientación: ahora, a cada uno de los puntos que se ha filtrado y clasificado como clave se les debe asignar una dirección, un gradiente en función de la zona que lo rodea (Figura 12).

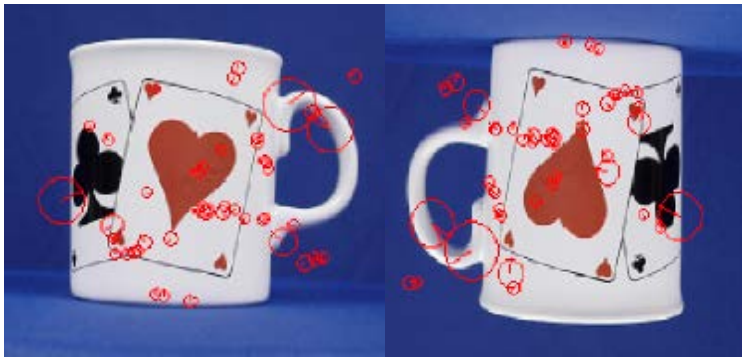

Figura 13. Imágenes con puntos clave y gradientes.

4) Descripción de puntos clave: Se debe aplicar un descriptor para que identifique la zona y para ello debe ser lo más invariable que podamos programar con el fin de no ver frustradas nuestra comparaciones por variaciones como la iluminación o el punto de vista (Figura 13).

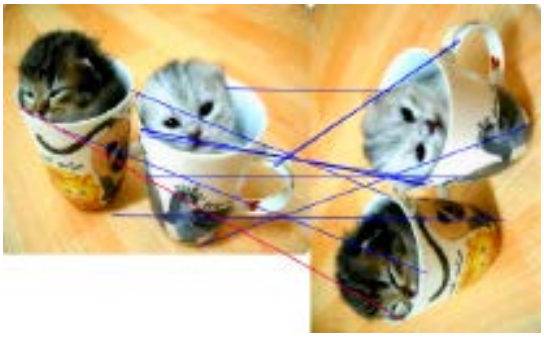

Figura 14. Localización de puntos iguales en imágenes con diferente orientación

3.2.1.2 Speeded-Up Robust Features (SURF)

Si tuviéramos que dar una definición rápida de este concepto no estaría muy desacertado decir que SURF es una evolución del SIFT aplicado a reconocimiento de objetos 3D.

Este detector de imágenes locales fue desarrollado por Herbert Bay en el 2006[12]. El motivo de que se aceptara tan positivamente este detector era su mayor fiabilidad y velocidad de análisis en comparación con su predecesor.

Para obtener las características de las imágenes y los puntos clave suma las respuestas obtenidas de las ondas Haar que ha aplicado sobre los mismos, pudiendo así a posteriori volver a calcular y reconstruir todo con ayuda de la imagen integral.

La secuencia del algoritmo SURF son tres: detección de los puntos de interés, descripción de los puntos de interés y concordancia de las características.

Como se puede apreciar la secuencia es muy parecida a las del SIFT, sin embargo la metodología del análisis del espacio escalado es diferente. En el caso del SURF lo que se hace es usar escalas variando el parámetro de escala σ siempre al doble del tamaño del núcleo. Además cada octava tiene varios subniveles con subdivisiones de menor escalamiento (Figura 14).

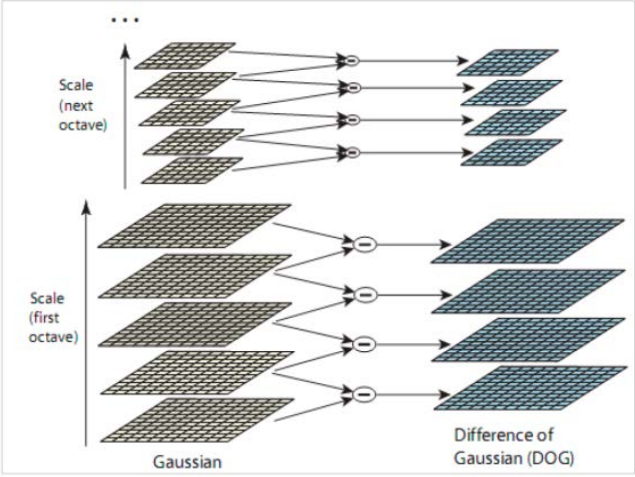

Figura 15. Por cada Octava en diferentes escalas se aplica una Diferencia Gaussiana

Así se estableció la siguiente ecuación para el escalado de muestreo:

$$\sigma = \frac{1,2}{3}\,(2^{o} * i + 1) = \frac{1,2}{3} * l \quad (3)$$

Siendo o el índice de la octava, i el índice del intervalo y $l/3$ el ancho de banda del filtro correspondiente.

También los recursos usados en la detección de los puntos de interés son diferentes. En este caso deberemos en primera instancia detectar las características a través del determinante normalizado-escalado de Hessian, el cual hace uso de la función Gaussiana y no de la Laplaciana (Figura 15).

$$D\bar{o}H^{(\sigma)}(u) \coloneqq \det(\bar{H}^{(\sigma)} * u = \frac{\left(\bar{D}_{xx}^{(\sigma)} * u\right)x\left(\bar{D}_{yy}^{(\sigma)} * u\right) - \left(r(\sigma)\bar{D}_{xy}^{(\sigma)} * u\right)^2}{l^4} \quad (4)$$

$$\text{Con} \quad r(\sigma) = \frac{\left|\left|\bar{D}_{xx}^{(\sigma)}\right|\right|_F}{\left|\left|\bar{D}_{xy}^{(\sigma)}\right|\right|_F} * \frac{\left|\left|\bar{D}_{xy}^{(\sigma)} g_\sigma\right|\right|_F}{\left|\left|\bar{D}_{xx}^{(\sigma)} g_\sigma\right|\right|_F} \quad (5)$$

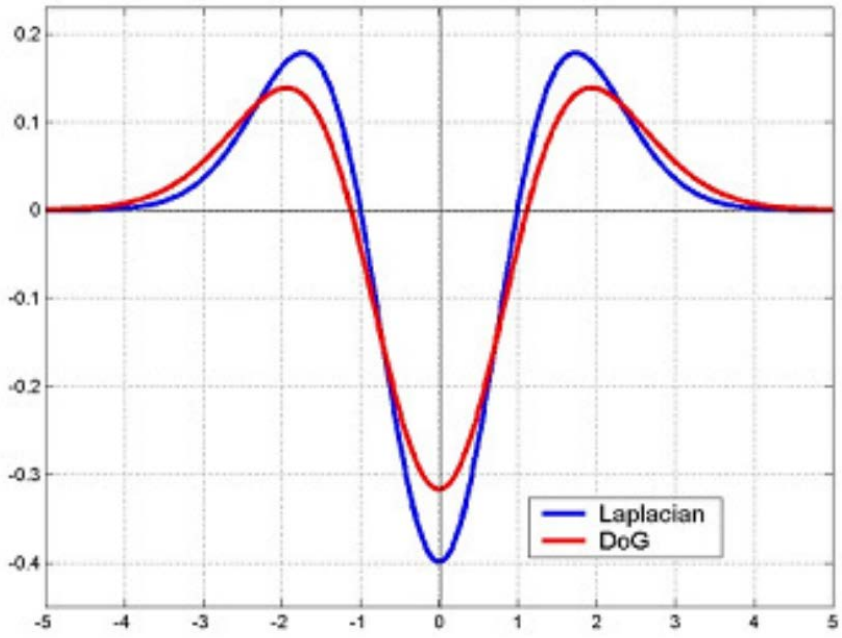

Figura 16. Diferencia entre la función Laplaciana y la Gaussiana

En segundo lugar seleccionaremos las características y para esos nos valdremos del método de la supresión no máxima y un umbral. Este método tiene en cuenta que los puntos de interés son covariantes con cualquier transformada similar y a además se corresponden al máximo local del operador DoH. Para evitar interferencia del ruido el algoritmo selecciona la característica saliente desde este conjunto de máximos locales. Para ello hacemos uso del umbral $t_H$en la respuesta del operador DoH para cada punto de interés candidato (x, y)en la escala σ.

$$\left|D\bar{o}H^{(\sigma)}(u)(x,y)\right| > \boldsymbol{t}_H \quad (6)$$

Por último procederemos a un refinamiento del espacio escalado a través de un esquema de diferencial finito. Para cada máximo local del operador DoH, la localización de los puntos de interés M y sus coordenadas (x, y y z) usaremos una interpolación simple para refinarlo. Así con la siguiente fórmula de entrenamiento cuadrático obtendremos resultados aún más óptimos:

$$M' = M + \delta \quad (7)$$

$$\text{Con } \delta = \begin{pmatrix} \delta_x \\ \delta_y \\ \delta_\sigma \end{pmatrix} = -H^{-1}\left(D\bar{o}H^{(\sigma)}(u)(x,y)\right) x \nabla \left(D\bar{o}H^{(\sigma)}(u)(x,y)\right) \quad (8)$$

Una vez explicado el proceso de análisis y extracción de características de los puntos de interés es preciso crear el descriptor local. Pero para eso se debe, como en el caso del SIFT, determinar una orientación dominante con la finalidad de conseguir una invarianza total a las rotaciones. Siguiendo las mismas pautas a SIFT la orientación local de una característica en un espacio de escala se calcula a partir de la distribución local de la orientación del gradiente. Así, para cada punto de interés se determina una región trazada con una circunferencia desde el punto de interés, dentro de la cual podremos calcular el gradiente por convolución con una serie de filtros. Como queremos evitar efectos secundarios del entorno todas las muestras de gradiente serán sometidas a una ponderación que usan un núcleo Gaussiano con una desviación estándar dada y la cual es dependiente de la distancia euclídea a la muestra del punto de interés correspondiente. Pero si el SIFT usaba histogramas para estimar la orientación dominante, el algoritmo SURF lo que hace es calcular el máximo de las puntuaciones teniendo en cuenta los ejes dominantes (Figura 16).

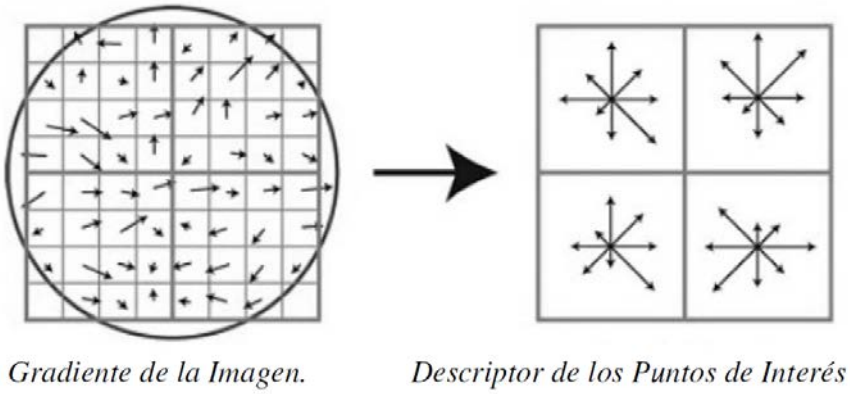

Figura 17. Descripción gráfica de la ventana gaussiana

Usando una ventana gaussiana se examinan la vecindad del punto de interés. Después sigue un tratamiento con histogramas para 8 orientaciones distintas.

El último paso sería el contraste de resultados. Para ello se cogen la pareja de imágenes a ser coincidentes y a través de sus puntos de interés se representa dos conjuntos. Lo que hacemos sería comparar los vectores a través de la técnica de umbralización denominada Distance Ratio (NN-DR) heredada de SIFT, cuya ventaja es que su dependencia de la dinamicidad de la imagen era menor que en el caso del umbral usado en SURF.

3.2.2 Extracción de características a partir de modelos 3D

Hasta ahora se ha visto cómo con una imagen se era capaz de extraer características. Pero con medios (por ejemplo CAD[13]) que captan la realidad en 3D también posible manipular esta información para realizar el mismo proceso con resultados más completos y productivos que con modelos 2D.

Hay diversos métodos que se usan para extraer características de estos modelos 3D, se va enumerar y explicar los más relevantes en el entorno de la investigación y del desarrollo de la visión por computadora.

3.2.2.1 Descriptores de color

En ocasiones se podrá tener un caso en el que la única diferencia entre los objetos son los colores de los mismos. Para estos casos, además de usar en un primer momento un descriptor de forma, se hace imprescindible aplicar posteriormente un descriptor de color que permita obtener sus características más relevantes para clasificarlo e identificarlo.

La Kinect recibe información a color a través de su cámara RGB. Esto quiere decir que a través de la composición de los 3 colores básicos (R-red-rojo, G-green-verdel y B-blue-azul) es capaz de obtener los mismos colores que definen en realidad los objetos y el entorno.

[13] En inglés Computer aided design

Así pues es posible, mediante un descriptor de color, crear un histograma de color tridimensional para cada punto o bien 3 histogramas de color de una única dimensión.

En ambos casos se deberá tener en cuenta que los histogramas representan colores absolutos, y no tienen en cuenta sombras o intensidades de colores. Sin embargo este incoveniente tiene fácil solución, ya que un sombrado o un color con diferente intensidad es simplemente el valor absoluto multiplicado por un escalar en las tres componentes RGB.

Es decir, buscamos un descriptor de color τ que satisfaga la siguiente condición:

$$\tau\left(\begin{pmatrix}R\\G\\B\end{pmatrix}\right) = \tau\left(\begin{pmatrix}\alpha & 0 & 0\\0 & \alpha & 0\\0 & 0 & \alpha\end{pmatrix}\begin{pmatrix}R\\G\\B\end{pmatrix}\right) \quad (9)$$

3.2.2.2 Descriptores de forma

Si investigamos un poco acerca de los descriptores de forma de modelos 3D nos daremos cuenta que su número es muy inferior a los basados en 2D, y es porque el mundo 3D es, sin lugar a duda, muy reciente, aún más si lo comparamos con el tiempo que llevamos manejando información de modelos 2D. Sin embargo, si tenemos en cuenta que una gran mayoría de los descriptores 3D están basados y desarrollados a partir de los 2D, es muy acertado decir que se tratan de algoritmos mucho más complejos pero a su vez más concretos, exactos y que proporcionan mejores resultados.

Al final lo que deseamos es que los descriptores nos arrojen más luz sobre las principales características geométricas de las imágenes procesadas con el fin de clasificarlas en un paso posterior con la mayor facilidad y fiabilidad posible.

A continuación vamos a explicar cómo funcionan los descriptores 3D. Para hacer una rápida introducción al lector diremos que todos los modelos 3D se analizan, como ya vimos, como conjunto de puntos de interés, y a su vez esos puntos nosotros los agruparemos en formas geométricas de sencilla manipulación y cálculo de espacios y medidas. Las formas geométricas más usadas serán la esférica y la cúbica. ¿Y cuándo se usará una forma u otra? En realidad es casi indiferente, la

única condición es que, en función del tipo de distancias métricas que usemos, aplicaremos una forma geométrica u otra.

A continuación explicaremos las distancias métricas que podemos usar.

3.2.2.3 Distancias métricas

Por lo general entendemos por medida de distancia a un indicador que nos proporciona la lejanía entre dos objetos. En concreto, la distancia física la tratamos como una longitud que se calcula a través de una línea recta trazada entre dos puntos espaciales.

Sin embargo éste es sólo un criterio por el que nosotros nos basamos. Por hacer una similitud, el tiempo se puede medir en el sistema sexagesimal o decimal. De igual forma en el mundo de la computación encontramos bases binarias, logarítmicas, etc…

Cambiando las "reglas" de medida y referencia hacemos uso de otros métodos de análisis de distancias métricas. A continuación veremos los más usados en el ámbito de los descriptores de forma de modelos 3D.

i. Distancia euclídea

Ésta es probablemente la más usada de las distancias métricas. La distancia euclídea se define como la longitud de un segmento definido por una línea recta entre dos puntos. Se calcula a través del teorema de Pytagoras (Figura 17).

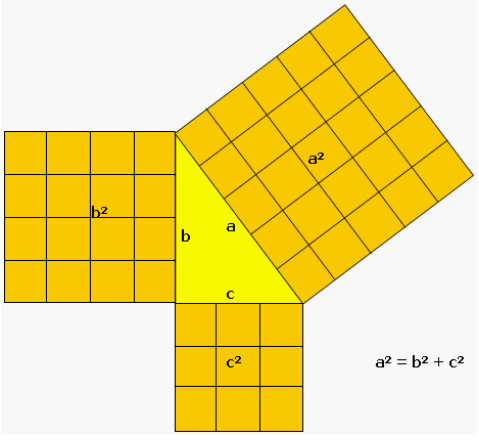

Figura 18. Teorema de Pytagoras

La siguiente fórmula nos proporciona la distancia euclídea entre dos puntos de n-dimensiones ambos, $\mathrm{P} = (\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3, \dots\ \mathrm{p}_\mathrm{n})$ y $\mathrm{Q} = (\mathrm{q}_1, \mathrm{q}_2, \mathrm{q}_3, \dots\ \mathrm{q}_\mathrm{n})$

$$l_2(P,Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \quad (10)$$

ii. Distancia ajedrecística o de Chebyshev

Se define como la distancia entre dos puntos calculada a través de la diferencia de coordenadas de los puntos implicados (Figura 18).

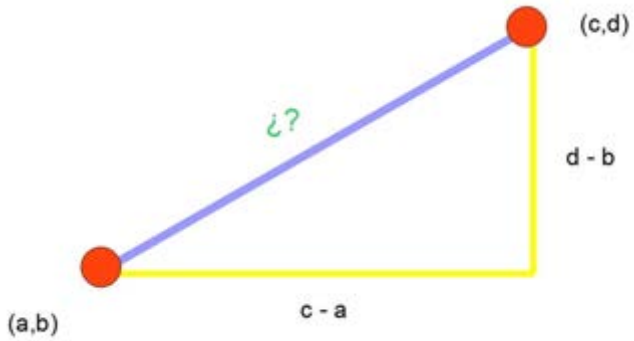

Figura 19. Problema trivial de distancia ajedrecística

Así pues en n dimensiones se define como la distancia ajedrecística entre dos puntos $\mathrm{P} = (\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3, \dots\ \mathrm{p}_\mathrm{n})$ y $\mathrm{Q} = (\mathrm{q}_1, \mathrm{q}_2, \mathrm{q}_3, \dots\ \mathrm{q}_\mathrm{n})$ como:

$$l_\infty(P,Q) = \max(|p_1 - q_1|, |p_2 - q_2|, \dots, |p_n - q_n|) \quad (11)$$

iii. Otras distancias

A continuación se presentan otras medidas relevantes a través de las cuales luego obtendremos características en base a éstas.

a) **Compactación**: a través del diámetro esférico equivalente (ESD) y del diámetro convexo o bien conocido el volumen y el área:

$$C_1 = \frac{ESD}{D_{ch}} \quad (12)$$

$$C_2 = \frac{(V^2 * 36 * \pi)}{S^3} \quad (13)$$

b) **Esfericidad**: Conocido el volumen y el área podemos determinar como de esférico es un objeto:

$$S = a * \frac{V^{2/3}}{S} \quad (14)$$

Siendo a = 4,84

c) **Elongación**: Es un indicador de forma resultante de la comparación ponderada del eje mayor y el eje menor del objeto. Es decir [ancho/largo], de ahí que su valor oscile entre 0 y 1.

d) **Convexidad**: Determina la aspereza de una superficie 3D. Se puede obtener usando el área convexa S_{ch} o bien el volumen convexo V_{ch}. La convexidad perfecta tiene un resultado unidad, sin embargo la superficie más irregular resultará un 0 en los cálculos:

$$C_s = \frac{S_{ch}}{S} \quad (15)$$

$$C_v = \frac{V_{ch}}{V} \quad (16)$$

3.2.2.4 Índices de forma

La explicación más sencilla de este concepto es la división de un objeto 3D en figuras geométricas que, a través de sus cualidades geométricas, nos permita manipular y analizar la figura que la compone.

Más estrictamente se define índices de forma como descriptores de forma que se define con parámetros, coeficientes o la combinación de éstos para otorgar información cuantitativa sobre la forma de un modelo 3D. Necesariamente los índices de forma deben ser adimensionales y no tener variación por rotación o translación.

Cuando se hace uso de la distancia euclídea hacemos una compactación de los puntos de interés usando la forma geométrica de una esfera. Siempre

consideraremos que las esferas no están vacías, es decir, que tienen un volumen positivo. Así pues también definiremos como dentro de cada esfera las coordenadas de cada punto de interés, usando radios (a poder ser) siempre iguales en todas las esferas.

Es por eso que a menor radio, mayor precisión de cálculo de distancias. Sin embargo esta condición implica mayor potencia de cálculo computacional y viene limitado por el número de puntos de interés, ya que, cuantos más puntos clave tengamos mayor número de esfera usaremos y de menor radio (o al menos se nos ofrece la posibilidad), pues podremos usar menos para aliviar el cálculo a costa de obtener un resultado menos exacto (Figura 19).

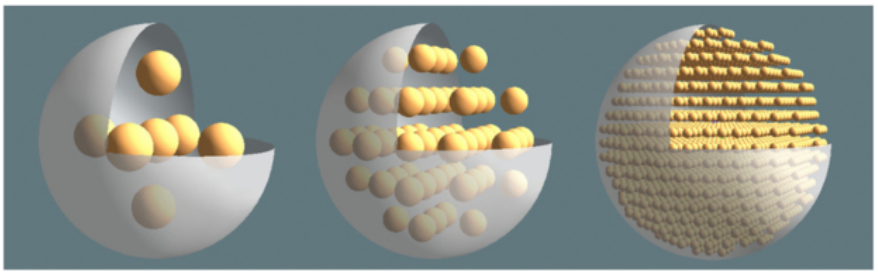

Figura 20. Ejemplo de precisión de una forma en función del nº de esferas y el radio de las mismas

Así pues, podremos calcular compacidad de cada una de las esferas $Q(t)$ siendo t un radio arbitrario positivo:

$$C_2^{3D}(Q(t)) = \frac{3^{8/3}}{5(4\pi)^{2/3}} * \frac{t^{2/3}}{2+t^2} \quad (17)$$

Ahora bien, si hacemos uso de la distancia ajedrecística compactaremos el modelo 3D con cubos (Figura 20). Las mismas conclusiones obtenidas en las esferas se pueden aplicar a los cubos (Figura 21).

Figura 21. Compactación por cubos de una silla en dos ejemplos con diferentes parámetros

En este caso definiremos la compacidad en función de los tres ejes de coordenadas y el radio r tomado:

$$C_{\infty}^{3D}(r*S) = \frac{3}{8} * \frac{(r^3\mu_{0,0,0}(S))^{4/3}}{r^4 * \min_{\emptyset,\theta\epsilon[0,2\pi]} \iiint_{R(S,\emptyset,\theta)} \max\{|x|,|y|,|z|\}dxdydz} \quad (18)$$

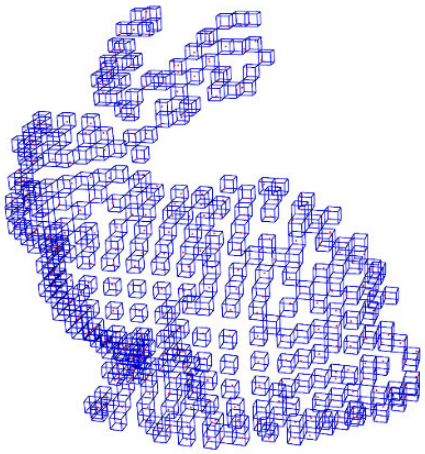

Figura 22. Descriptor de formas cúbicas aplicado sobre un conejo

3.2.3 Viewpoint Feature Histogram (VFH)

Se trata de un algoritmo desarrollado a partir de otro anterior denominado “Point Feature Histogram” (PFH). Por ello se va a definir brevemente antes el mismo para comprender el VFH.

El PFH es un algoritmo que permite representar en un histograma los valores obtenidos al comparar los ángulos de inclinación y oscilación de dos pares de puntos. Esto es posible porque conocemos las normales de ambos puntos.

$$u = n_s \quad (19)$$

$$v = u\ x\ \frac{(p_t - p_s)}{\|p_t - p_s\|_2} \quad (20)$$

$$w = u\ x\ v \quad (21)$$

Luego, la diferencia entre las normales n_s y n_t será:

$$\alpha = v * n_t \quad (22)$$

$$d = \|p_t - p_s\|_2 \quad (23)$$

$$\phi = \tan^{-1}\left(\frac{w * n_t}{u * n_t}\right) \quad (24)$$

Así reducimos los 12 valores iniciales de ambos puntos a tan solo 4, como muestra la Figura 22.

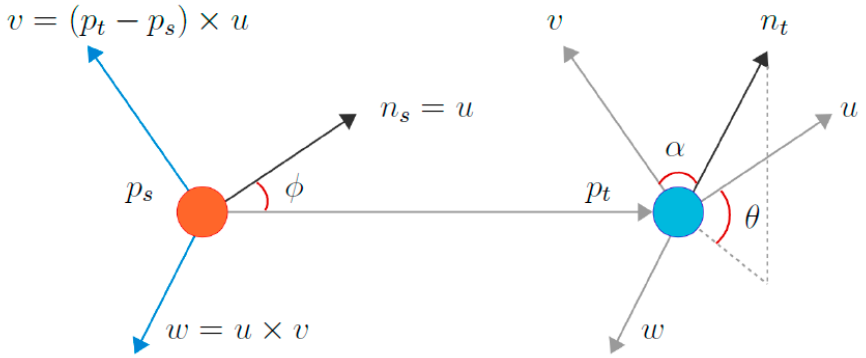

Figura 23. Representación de las variables que intervienen en el algoritmo PFH

Entre el PFH y el VFH se desarrolló otro algoritmo intermedio, pero en realidad se trata del PFH con un dato más, que es el punto de vista. Esto permite mejores resultados y más completos. Este algoritmo recibe el nombre de Fast Pointview Feature Histogram (FPFH) (Figura 23).

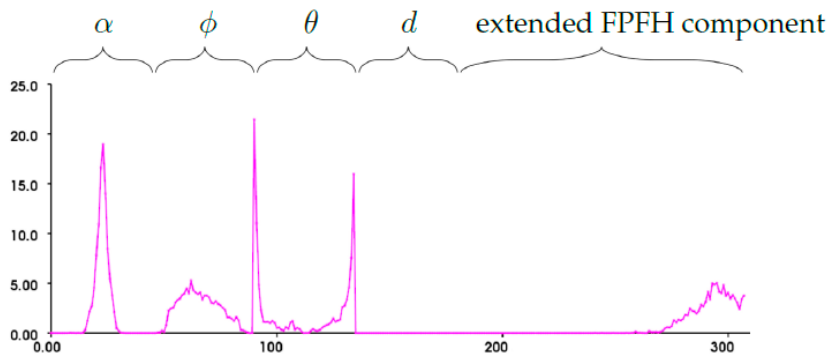

Figura 24. Histograma de un FPFH

Finalmente, el VFH (Figura 25) añade la variabilidad del punto de vista del FPVH a través del vector de dirección del mismo. El proceso completo es el siguiente:

a) Encontrar el centroide, c, del punto

b) Para cada punto ,p , de una nube de puntos se deberá establecer referencias locales (u, v, w), las cuales vienen definidas por las mismas ecuaciones que las del algoritmo PFH.

c) Encontrar los 3 ángulos (α, ϕ, θ) del PFH.

d) Añadir la componente del punto de vista que se obtiene como el resultado de representar en un histograma los ángulos entre cada dirección del punto de vista de cada punto de la nube y su correspondiente normal.

e) Sobre la dirección del punto de vista del centroide y cada una de las normales de la superficie, calculamos la componente que mide el giro relativo así como los ángulos de inclinación y oscilación (Figura 24).

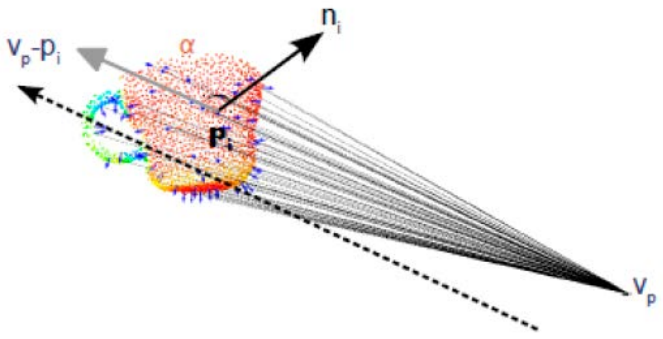

Figura 25. Representación geométrica del VFH

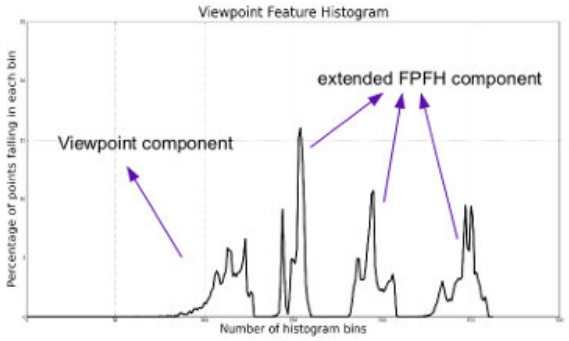

Figura 26. Ejemplo de histograma del VFH

4. RESULTADOS EXPERIMENTALES

El entorno de trabajo de las pruebas de trabajo ha sido, como se observa en la Figura 26, de una mesa con 3 objetos de diferentes tamaños sobre la misma.

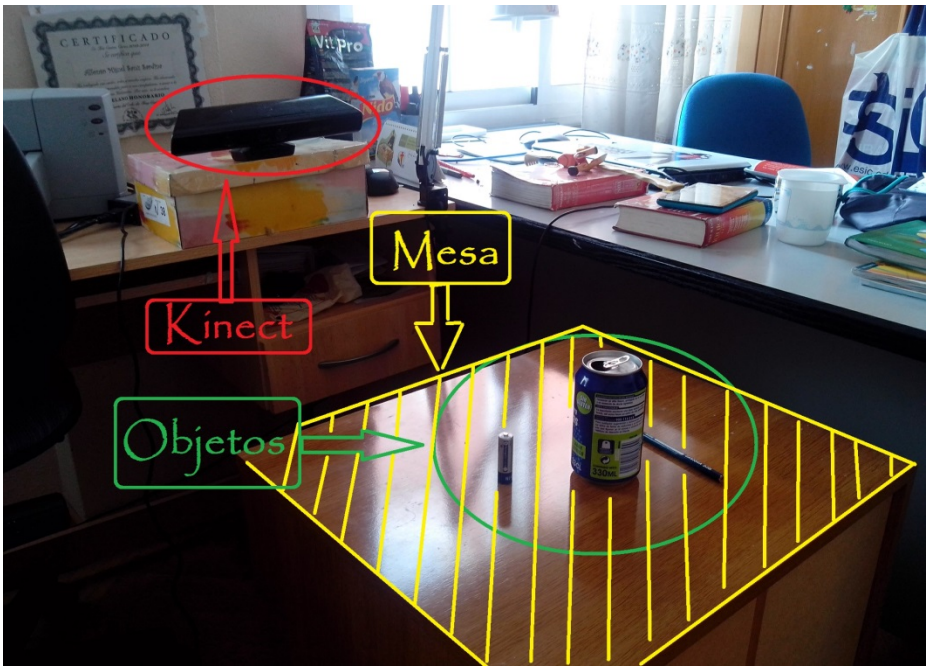

Figura 27. Entorno de trabajo

La Figura 27 muestra qué imagen se obtiene con la cámara Kinect para este entorno.

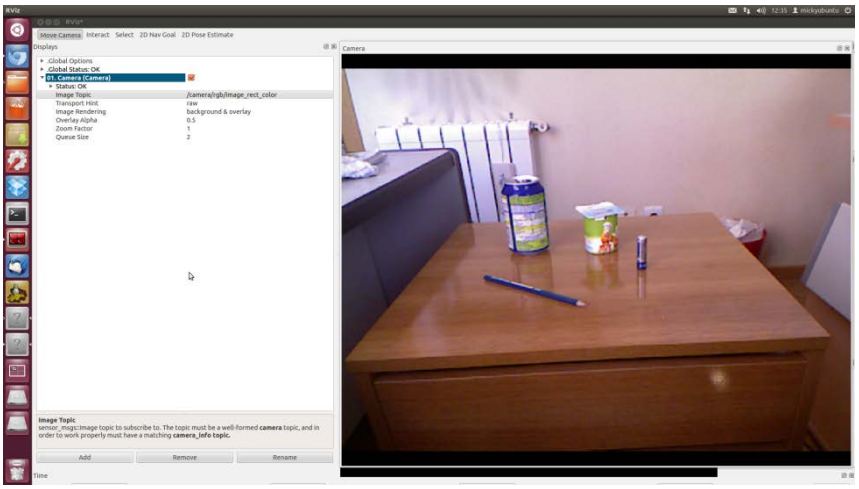

Figura 28. Perspectiva del entorno con la Kinect (puntos)

A continuación se detallará cual ha sido el procedimiento para la obtención de los datos experimentales, recogidos en el diagrama de flujo de la Figura 28.

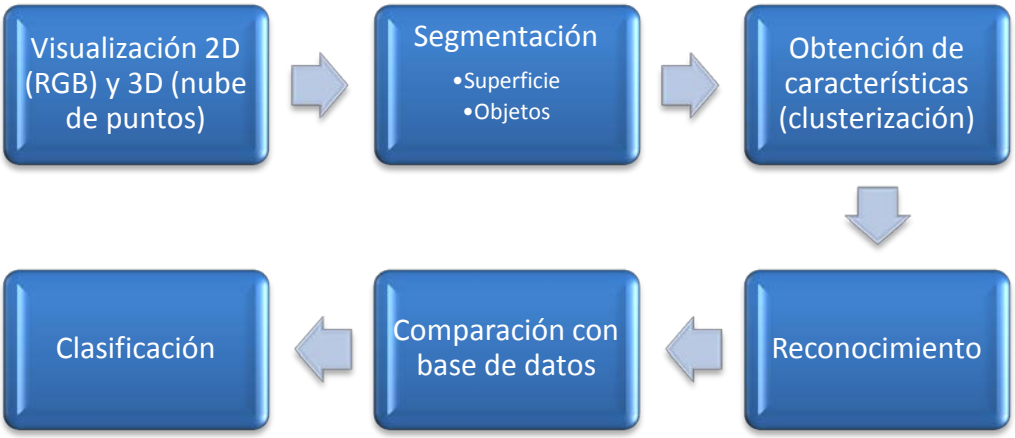

Figura 29. Procedimiento de adquisición de datos

4.1 Visualización

La primera fase de la experimentación es la visualización del entorno. Para ello sólo ha intervenido la cámara Kinect y el programa de visualización rviz (incluido en ROS fuerte).

Las imágenes que se pueden obtener varían en función de las propiedades que se desea que muestre la Kinect, siendo posible una imagen RGB (Figura 27) y monocolor –Black&White- (Figura 29).

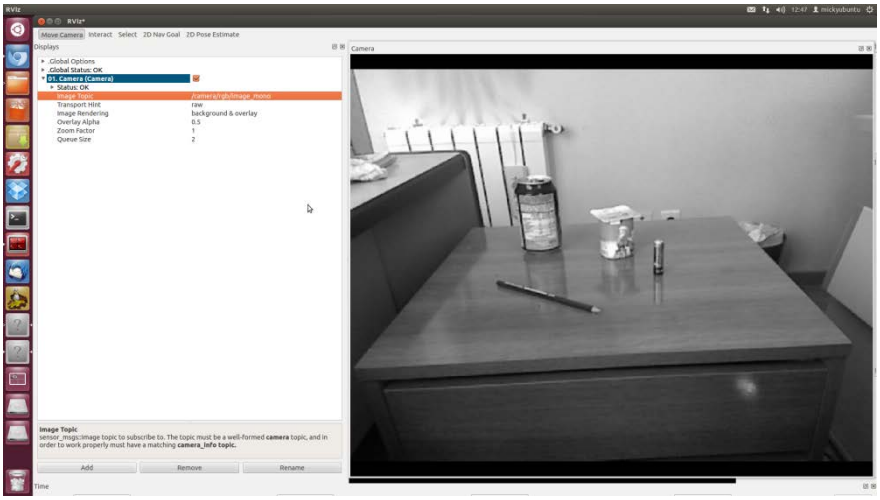

Figura 30. Imagen Monocolor (B&W)

Además también es posible determinar cuál es el patrón de representación de los datos adquiridos, siendo posible elegir entre los tipos "puntos" (Figura 27), "esferas" –de radio ajustable- (Figura 30 –radio 0.01- , Figura 31 –radio 0.001- y Figura 32 – radio 0.05-) y "Cubos" – Box- (Figura 33).

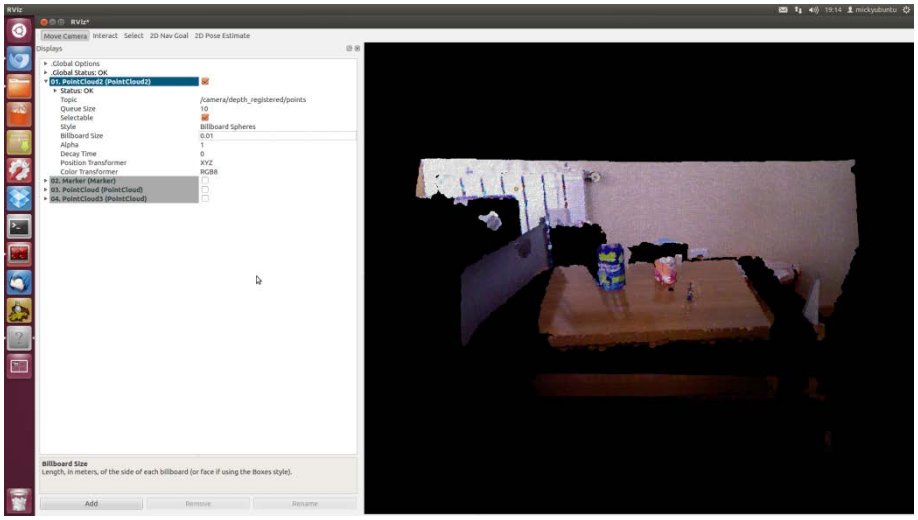

Figura 31. Imagen RGB con radio de esferas 0.01

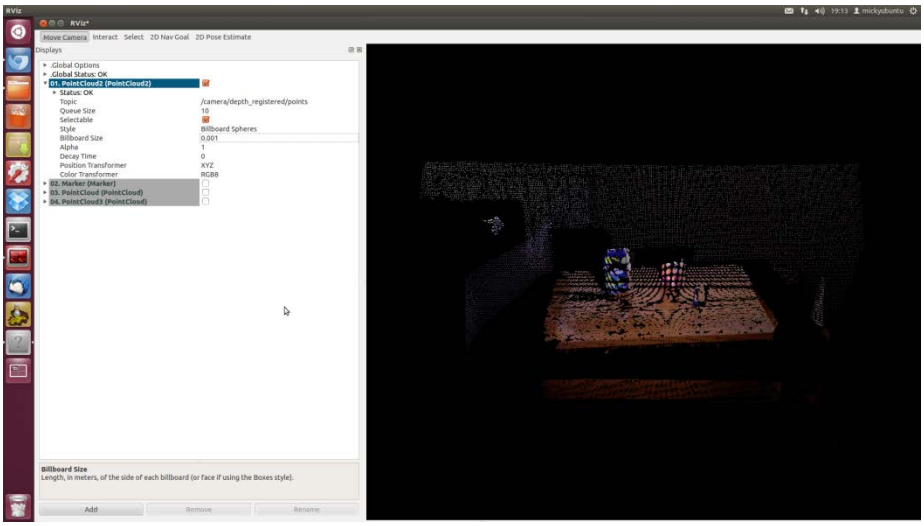

Figura 32. Imagen RGB con radio de esferas 0.001

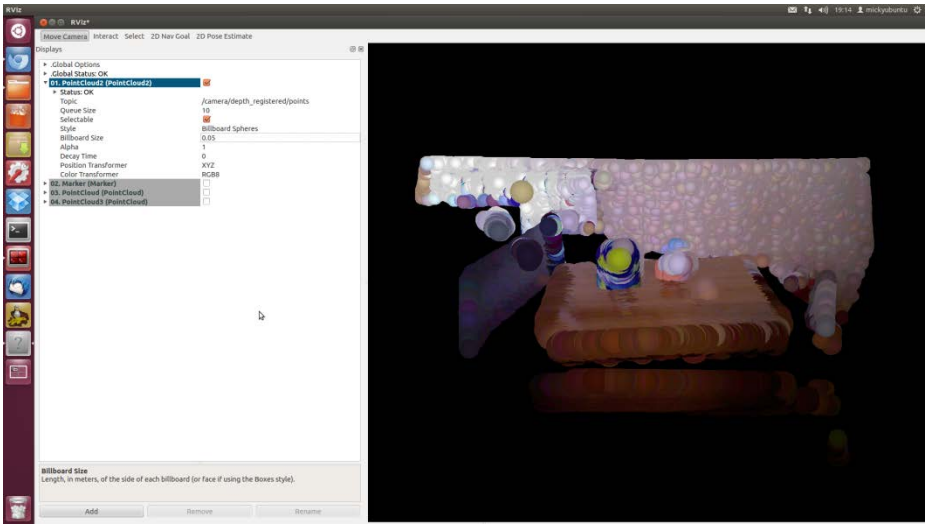

Figura 33. Imagen RGB con radio de esferas 0.05

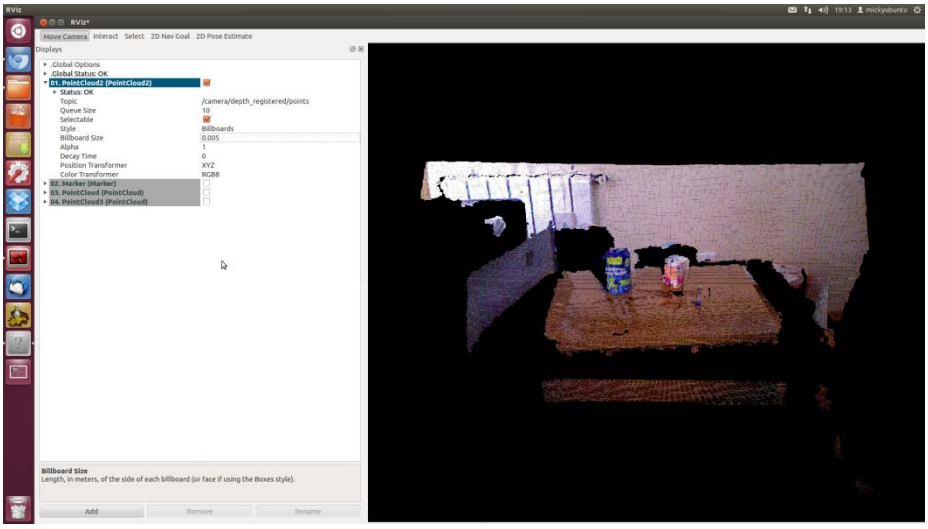

Figura 34. Imagen RGB dibujada con cubos

Por último se puede en todos los casos cambiar la perspectiva ya que siempre la imagen se presenta en 3D, posibilitando diferentes perspectivas[14] (Figura 34 y Figura 35).

[14] La circunferencia amarilla indica la posición de la Kinect

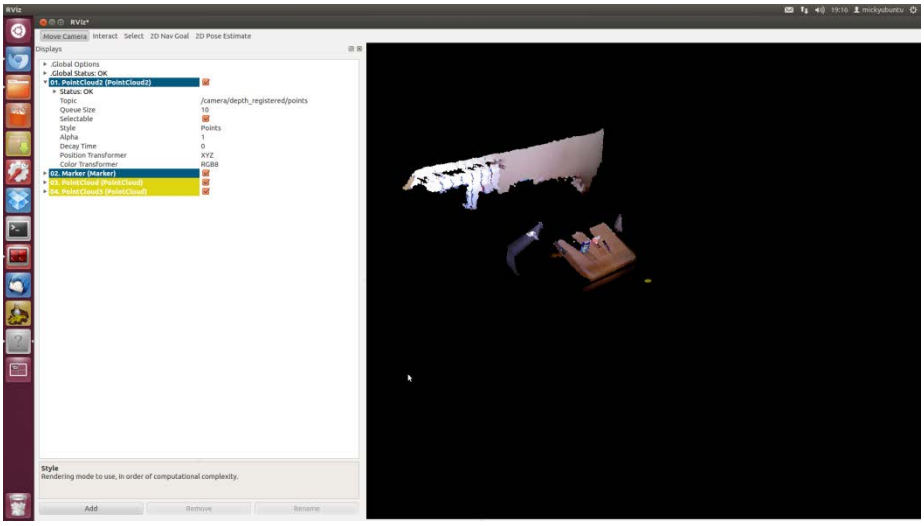

Figura 35. Perspectiva superior izquierda

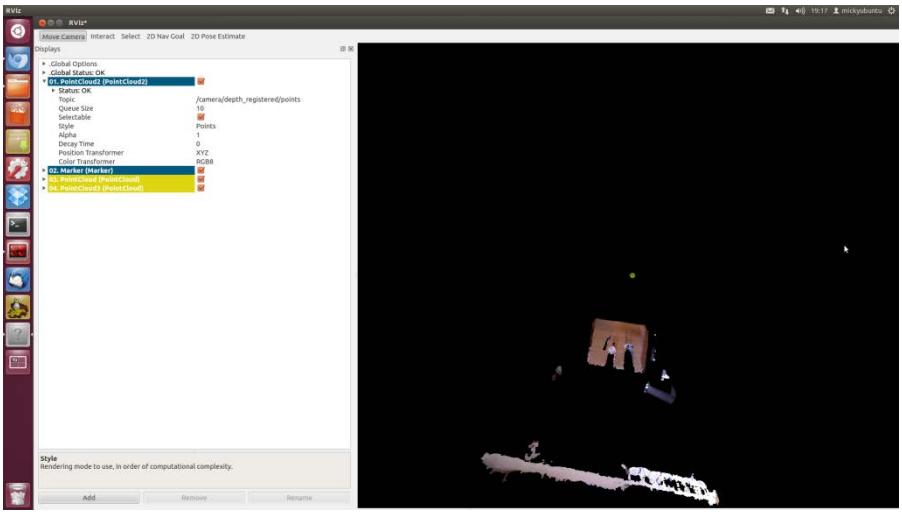

Figura 36. Perspectiva en planta

4.2 Segmentación

En esta fase el objetivo es detectar la superficie plana sobre la que reposan los objetos así como los mismos. En este caso se hace posible gracias al stack uc3m_extractObjTable encargado de la segmentación de la superficie y los objetos. A continuación se segmentará la mesa de la Figura 36.

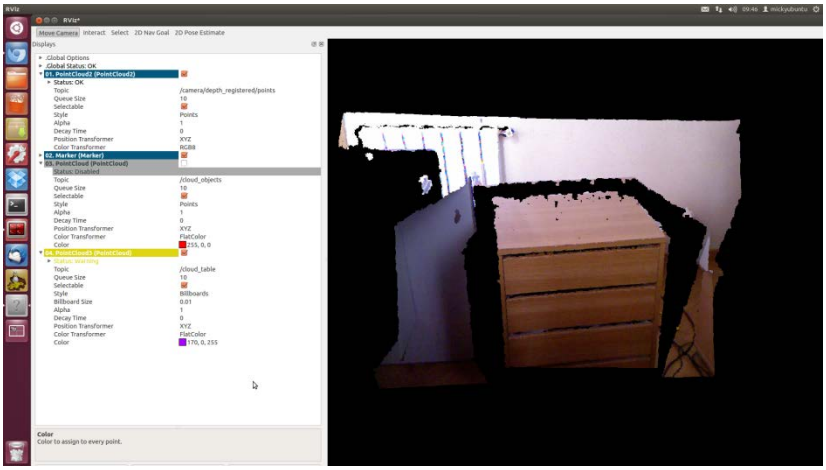

Figura 37. Imagen de color de la mesa a segmentar

A continuación se lanza el script del stack para que proceda a la segmentación de la superficie. En la Figura 37 se aprecia como la línea amarilla ha trazado un cuadrilátero que representa el plano al que pertenece la mesa y los cuadrados morados son los puntos que ha identificado como parte de la superficie plana de la misma.

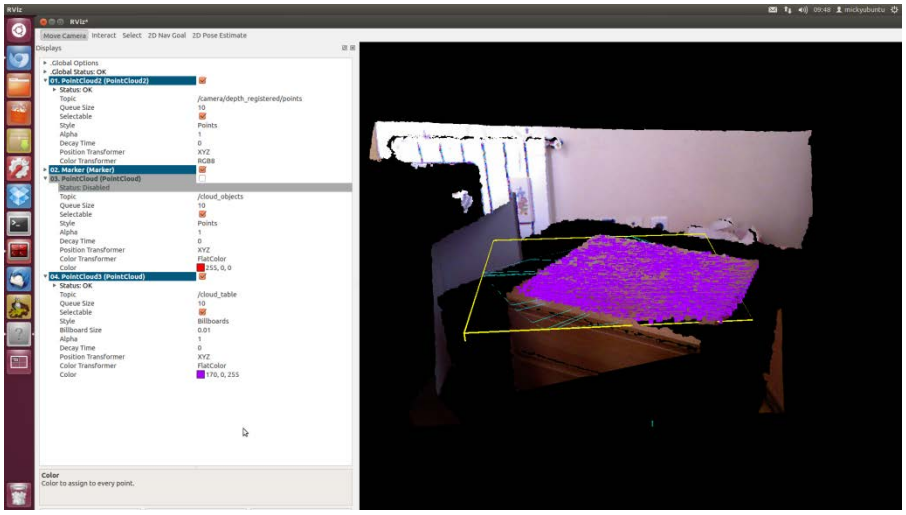

Figura 38. Imagen de color de la mesa segmentada

Por supuesto es posible (y para la fase de reconocimiento será necesario) eliminar toda la información que no pertenece a la mesa (Figura 38).

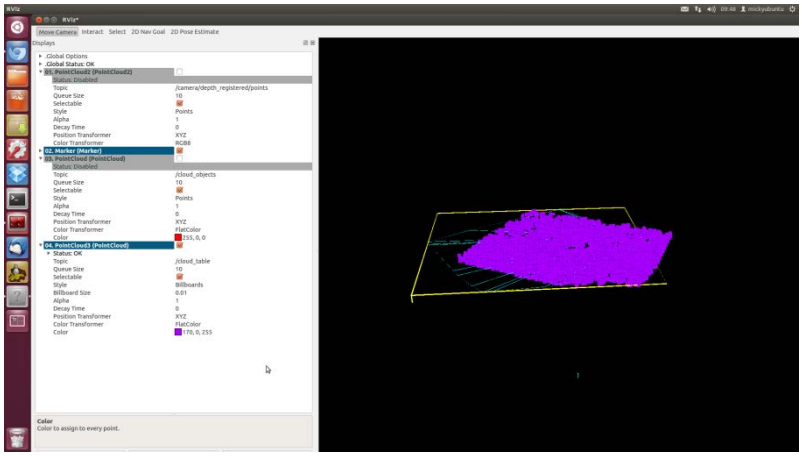

Figura 39. Mesa filtrada

El algoritmo principal se basa, como se describió en el apartado 3.1.1 (punto 2), en la ecuación de una superficie plana. A continuación se muestra en la tabla 1 la implementación principal de este concepto en la programación del código de este stack.

```
/**
 * This program is free software: you can redistribute it
and/or modify
 * it under the terms of the GNU Lesser General Public
License as published by
 * the Free Software Foundation, either version 3 of the
License, or
 * (at your option) any later version.
 *
 * Author: Silvia Rodríguez-Jiménez <srjimene@ing.uc3m.es>,
(C) 2012
 */

/*! Create a marker for a convex hull table
  It is the responsibility of the caller to set the
appropriate pose for the marker so that
  it shows up in the right reference frame.
 */

// Assumes plane coefficients are of the form ax+by+cz+d=0,
normalized
tf::Transform getPlaneTransform (pcl::ModelCoefficients
coeffs, double up_direction, bool flatten_plane)
{
      ROS_ASSERT(coeffs.values.size() > 3);
      double a = coeffs.values[0], b = coeffs.values[1], c =
coeffs.values[2], d = coeffs.values[3];
      double aux = a;
      //[ INFO] [1351271918.366466271]: p0x:0.204942 p0y:-
0.201516 p0z:-0.582959
      //[ INFO] [1351271918.366722170]: p0x:0.582959 p0y:-
0.204942 p0z:-0.201516
      a = -c; c = b; b = -aux;
```

```
      //asume plane coefficients are normalized
      btVector3 position(-a*d, -b*d, -c*d);
      btVector3 z(a, b, c);

      //if we are flattening the plane, make z just be
(0,0,up_direction)
      if(flatten_plane)
      {
        ROS_INFO("flattening plane");
        z[0] = z[1] = 0;
        z[2] = up_direction;
      }
      else
      {
        //make sure z points "up"
        ROS_INFO("in getPlaneTransform, z: %0.3f, %0.3f,
%0.3f", z[0], z[1], z[2]);
        if ( z.dot( btVector3(0, 0, up_direction) ) < 0)
        {
          z = -1.0 * z;
          ROS_DEBUG("flipped z");
        }
      }

      //try to align the x axis with the x axis of the
original frame
      //or the y axis if z and x are too close too each
other
      btVector3 x(1, 0, 0);
      if ( fabs(z.dot(x)) > 1.0 - 1.0e-4) x = btVector3(0,
1, 0);
      btVector3 y = z.cross(x).normalized();
      x = y.cross(z).normalized();

      btMatrix3x3 rotation;
      rotation[0] = x;      // x
      rotation[1] = y;      // y
      rotation[2] = z;      // z
      rotation = rotation.transpose();
      btQuaternion orientation;
      rotation.getRotation(orientation);
      ROS_DEBUG("in getPlaneTransform, x: %0.3f, %0.3f,
%0.3f", x[0], x[1], x[2]);
      ROS_DEBUG("in getPlaneTransform, y: %0.3f, %0.3f,
%0.3f", y[0], y[1], y[2]);
      ROS_DEBUG("in getPlaneTransform, z: %0.3f, %0.3f,
%0.3f", z[0], z[1], z[2]);
      return tf::Transform(orientation, position);
}
```

Tabla 1. Algoritmo principal programado de segmentación de mesa

Para detectar los objetos se hace uso de la técnica de anulación del fondo (background) y de los puntos previamente clasificados como parte de la mesa o superficie (filtrado).

Para demostrar que no importa el tamaño o el color de la mesa se expone de la Figura 39 a la Figura 41 otra mesa como ejemplo.

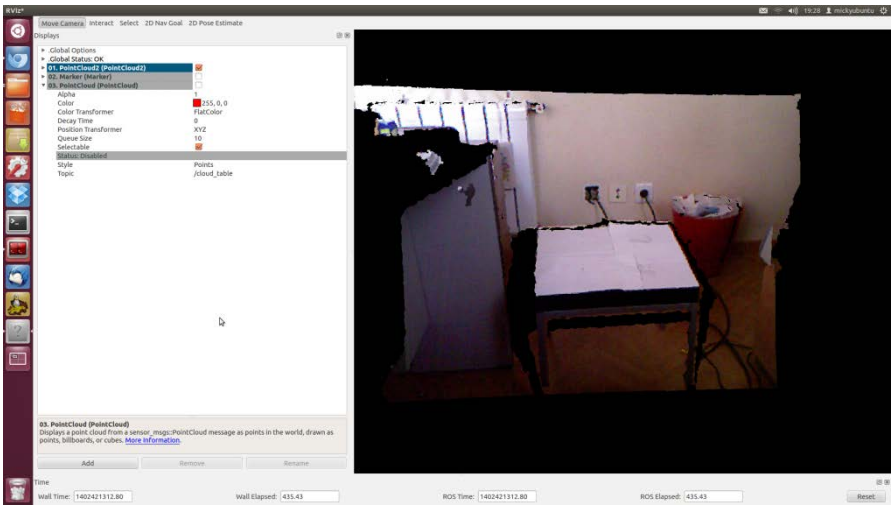

Figura 40.Mesa 2 RGB

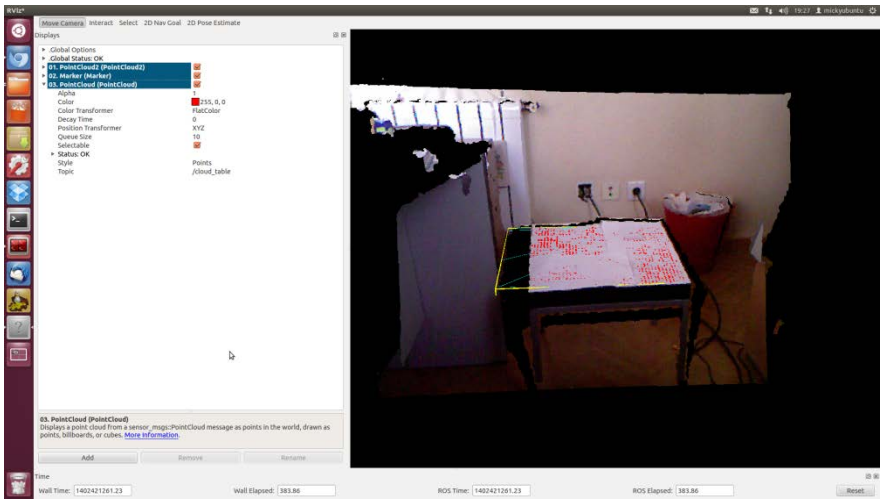

Figura 41. Mesa 2 Segmentada en RGB

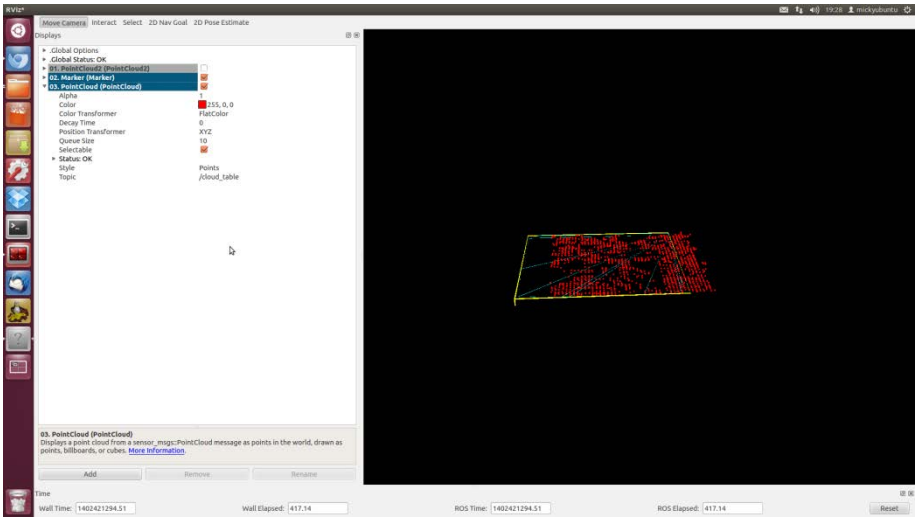

Figura 42. Mesa 2 Segmentada y filtrada

Una vez segmentada (y filtrada) la mesa, el stack es capaz de segmentar los objetos que reposan encima. En la figura 42 se puede observar la detección de objetos (formados por una nube de puntos blanca).

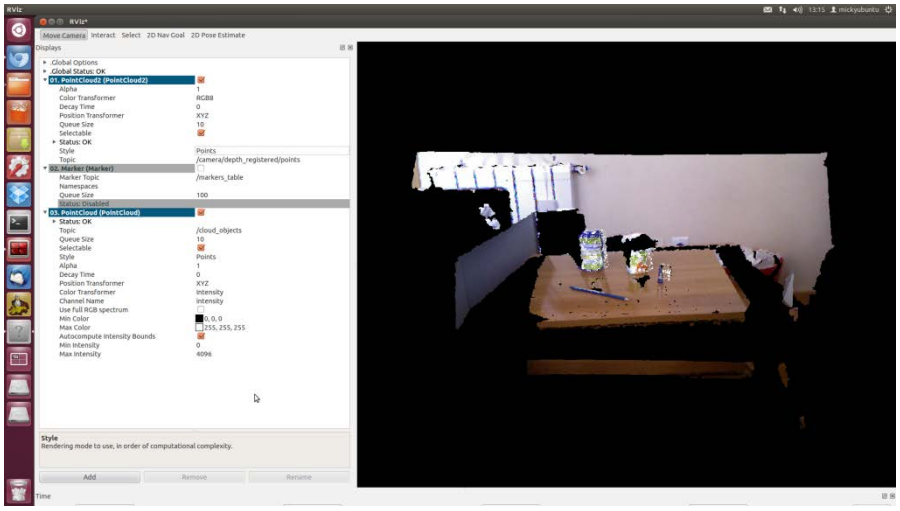

Figura 43. Segmentación de objetos

Y si filtramos la imagen para que sólo queden a la vista los objetos obtenemos la imagen de la Figura 43, apta para aplicarle el paso de reconocimiento.

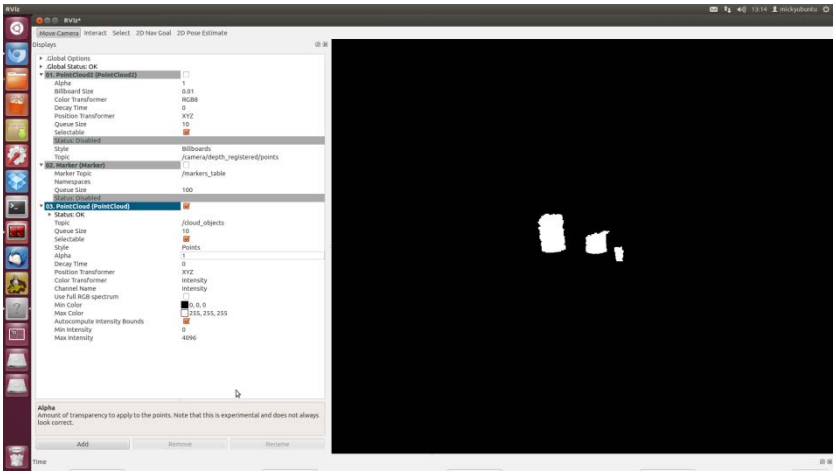

Figura 44. Segmentación de objetos listos para reconocimiento

Las principales líneas de código que hacen esto posible son las expuestas en la tabla 2 y la tabla 3, encargadas respectivamente de la adquisición de clústeres (características) procedentes de imagen RGB (2D) e imagen de profundidad (3D).

```
/**
 * This program is free software: you can redistribute it
and/or modify
 * it under the terms of the GNU Lesser General Public
License as published by
 * the Free Software Foundation, either version 3 of the
License, or
 * (at your option) any later version.
 *
 * Author: Silvia Rodríguez-Jiménez <srjimene@ing.uc3m.es>,
(C) 2012
 */


TableTopObjectDetector (ros::NodeHandle nh)
            : grabber_(nh), nh_(nh)
        {
            ros::NodeHandle param_nh("~");
            param_nh.param<double>("up_direction",
up_direction_, -1.0);
            param_nh.param<std::string>("processing_frame",
processing_frame_, "/camera_rgb_frame");
            param_nh.param<std::string>("world_frame",
world_frame_, "/world");
            param_nh.param<double>("object_voxel_size",
object_voxel_size_, 0.003);
            param_nh.param<double>("background_voxel_size",
background_voxel_size_, 0.01);
            param_nh.param<double>("depth_min", depth_min_,
-1.6);
```

```
            param_nh.param<double>("depth_max", depth_max_,
-0.4);
            param_nh.param<double>("object_height_min",
object_height_min_, 0.01);
            param_nh.param<double>("object_height_max",
object_height_max_, 0.5);
            param_nh.param<double>("max_distance_to_plane",
max_distance_to_plane_, 0.03);
            param_nh.param("path_current_frame",
path_current_frame_, string(" "));
            param_nh.param<bool>("use_image", use_image_,
true);
            param_nh.param<bool>("camera_aligned_verticaly",
camera_aligned_vertically_, true);

            cloud_objects_ =
nh_.advertise<sensor_msgs::PointCloud> ("cloud_objects",
10);
            cloud_table_ =
nh_.advertise<sensor_msgs::PointCloud> ("cloud_table", 10);
            marker_table_ =
nh_.advertise<visualization_msgs::Marker>("markers_table",
10);

            // ---[ Create all PCL objects and set their
parameters

detector_.setObjectVoxelSize(object_voxel_size_);

detector_.setBackgroundVoxelSize(background_voxel_size_);
            detector_.setDepthLimits(depth_min_,
depth_max_);

detector_.setObjectHeightLimits(object_height_min_,
object_height_max_);

detector_.setMaxDistToPlane(max_distance_to_plane_);
            detector_.initialize();

            if(grabber_.initialize())
            {
                ROS_ERROR("Grabber failed to initilize");
                return;
            }
            grabber_.start();
        }
```

Tabla 2. Algoritmo principal programado de segmentación de objetos

En la Figura 44 se muestran los principales pasos que tienen lugar al realizar la clusterización 3D.

Figura 45. Clusterización 3D

Por último se presentan de la Figura 45 a la Figura 49 la segmentación simultánea de la mesa y los objetos.

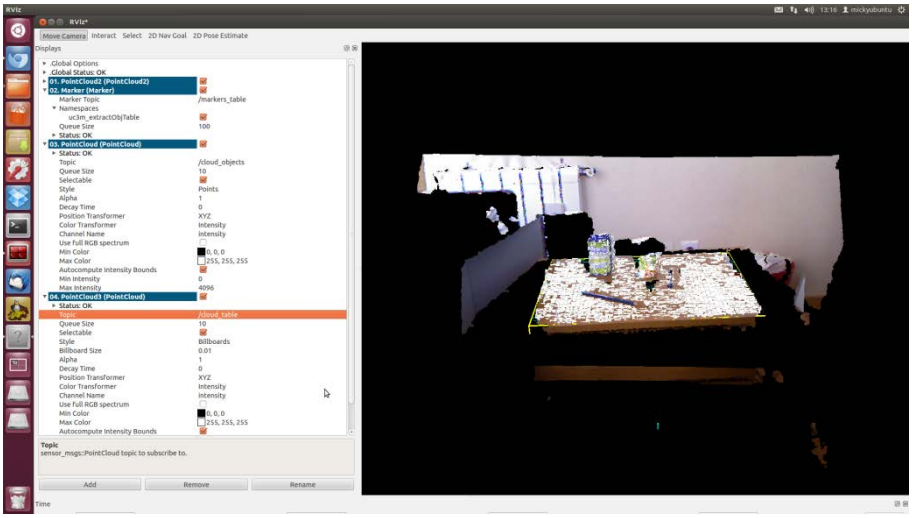

Figura 46. Segmentación de mesa y objetos RGB

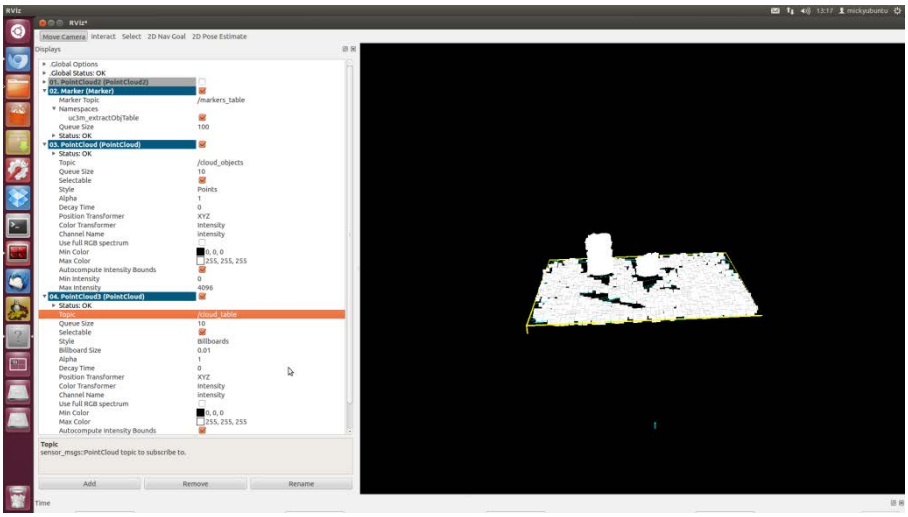

Figura 47. Segmentación de mesa y objetos RGB y filtrada

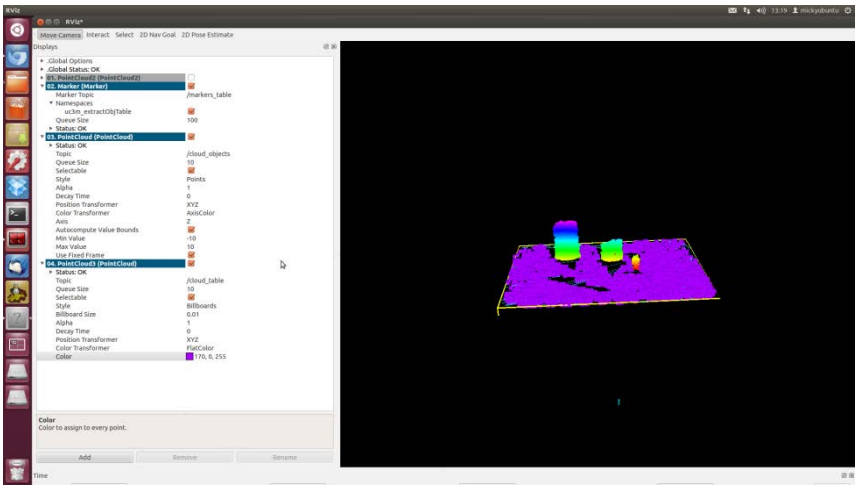

Figura 48. Segmentación de mesa y objetos RGB y filtrada con contraste

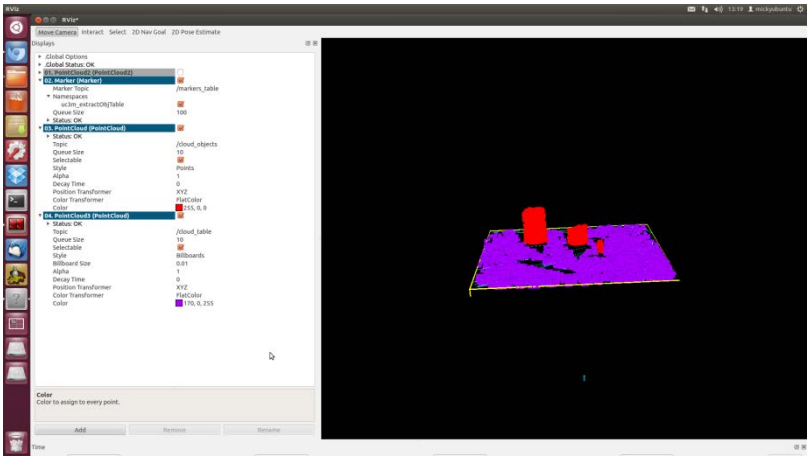

Figura 49. Segmentación de mesa y objetos RGB y filtrada con contraste en rojo

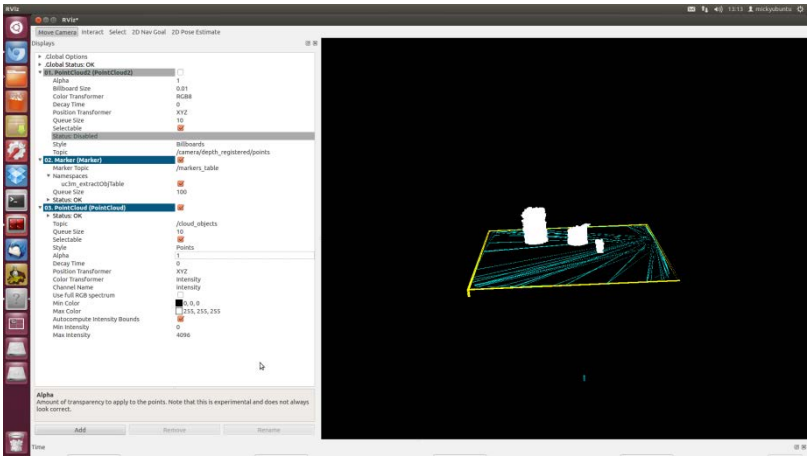

Figura 50. Segmentación de objetos y contorno de mesa

5. CONCLUSIONES

Una vez finalizado el tiempo disponible para realizar la recolección de datos experimentales se va a proceder a comparar cuales de los objetivos, enumerados en el punto 1.2 de esta memoria, han sido cumplidos satisfactoriamente.

El objetivo principal del proyecto era el reconocimiento 3D de objetos con una cámara Kinect sacando provecho de su potencial. Sin duda este punto ha sido cumplido casi en su totalidad pues se ha logrado la visualización, segmentación e identificación de objetos 3D, quedando pendiente en trabajo futuro la clasificación.

El segundo objetivo y más ambicioso era el de incorporar una base externa de datos de objetos 3D. Este punto claramente ha sido imposible abordarlo por falta de tiempo y es el punto fuerte del que se hablará en trabajo futuro.

De los objetivos detallados se puede concluir:

1. Se ha logrado muy satisfactoriamente el manejo y dominio del sistema operativo ROS, sus herramientas y sus posibilidades dentro de la versión ROS Fuerte en relación al reconocimiento de objetos.

2. Se ha incorporado todos los stacks de uc3m_handle para la versión fuerte, así como otros paquetes (nestk) de versiones anteriores (ROS electric) y además librerías y stacks necesarios para el correcto funcionamiento de los stacks uc3m_handle. Es por eso que se concluye que el apartado de incorporación de stacks ha sido superado.

3. Se ha realizado identificación por dos métodos que permite la cámara Kinect así pues se concluye que el objetivo de usar las principales técnicas de reconocimiento que permite la Kinect ha sido cumplida.

4. En las pruebas experimentales se ha trabajado con 3 objetos de diferente forma y tamaño con buenos resultados. Por eso este punto de trabajo con objetos de simetría sencilla concluye cumplido.

5. y 6. Tanto la incorporación de una base externa de cualquier tipo como el aceleramiento de la clasificación con bases de datos grandes quedan para trabajo futuro.

También debe quedar en constancia en este apartado de conclusiones otros objetivos colaterales realizados satisfactoriamente los cuales son los que más conocimiento y crecimiento intelectual han aportado:

7. Destreza para uso de Ubuntu tanto a nivel GUI como por comandos a través del terminal.

8. Conocimiento sobre la visión 3D en general y el tratamiento de imágenes.

9. Inclusión, modificación y eliminación de librerías, archivos y paquetes en Ubuntu bien sea por necesidad, por corrupción y pérdida o por incompatibilidad de cualquier tipo.

10. Persistencia y autonomía en resolución de numerosos problemas tanto aislados como individuales.

11. Horas de documentación para comprensión de PC de sobremesa actuales para posteriormente montaje de uno.

12. Experiencia para tomar un trabajo realizado por otro grupo de personas y abordarlo desde cero de forma individual y sin conocimientos previos.

A continuación expondré mi opinión personal acerca del PFG en primera persona:

El primer reto que me planteé con este proyecto era el de llevar a cabo un trabajo todo lo autónomamente posible sobre algo ya trabajado que me gustase. En este TFG se aúna conocimientos de visualización por cámara RGB-D (cámara Kinect) y muchísimo más conocimiento de programación.

Es este último punto del que más he aprendido, porque quien ha programado sabe que la cantidad de errores que se manejan son muy grandes. Y aún más cuando se trata se manipula una programación de otras personas sorbe las cuales no hay tiempo físico para conocer todo el código, tipo de estructura de ejecución, lenguajes de programación apoyo para ciertos módulos, etc.

La parte “bonita” del proyecto es la que se puede observar en pantallazos en la parte experimental, pero detrás de eso hay casi 500 horas de trabajo sólo y exclusivamente en reparación de errores de instalación, compatibilización y

dependencias.

A nivel personal la parte que más me ha satisfecho ha sido la de llevar a cabo un proyecto de considerable envergadura (dados mis nulos conocimientos al comienzo del mismo) desde mi casa, con mis medios y únicamente mi persistencia y autonomía como herramienta de aprendizaje. Mi tutor y mi directora de proyecto respetaron esta metodología y contribuyeron siempre con consejos y pistas cuando tenía problemas pero no dando soluciones directamente ni abusando de atención hacia mí, sino dándome tiempo y ánimos. Sin lugar a dudas sería este último párrafo el que yo más destacaría del TFG.

6. FUTURO TRABAJO

Como se ha hecho mención en el apartado de conclusiones hay varios aspectos que quedaron pendientes de abordar parcial o completamente y que se presentan como trabajo futuro. Además se añade otro más para dejar varias vías de trabajo abiertas a este TFG. A continuación se enumeran:

Completar el último paso de un sistema de reconocimiento por cámara RGB-D que es la clasificación.

Manipulación de nuevas bases de datos, acondicionamiento de bases externas para uso exclusivo en este TFG y por último el diseño de un nuevo concepto de base de datos que permita más universalidad entre proyectos.

Creación de un script sencillo que automatice todos los pasos de visualización, clasificación y control de base de datos.

Migración del trabajo a la última versión de ROS que ofrece nuevas herramientas de visualización y segmentación así como mayor velocidad de ejecución, compilación e intuitividad en su uso independientemente del tipo del proyecto que se desarrolle.

7. BIBLIOGRAFÍA

[1] DiBona C.; Ockman S.; Stone M. "OPENSOURCES: Voices from the OpenSource Revolution" [en linea]. [consulta 30-3-2014]. Disponible en web: <http://www.smaldone.com.ar/documentos/libros/opensources.pdf>

[2] "Código abierto" [en línea]. Wikipedia.org. [consulta 22-3-2014]. Disponible en web: <http://es.wikipedia.org/wiki/C%C3%B3digo_fuente>

[3] "Kinect for Xbox 360" is Official Name of Microsoft's Controller-Free Game Device" [en línea]. Microsoft. [consulta 18-3-2014]. Disponible en web: <http://www.microsoft.com/en-us/news/features/2010/jun10/06-13kinectintroduced.aspx>

[4] Widenhofer, B. "Inside Xbox 360's Kinect controller" [en línea]. Septiembre 2010. Revista digital EETimes. [consulta 15-3-2014]. Disponible en web: < http://www.eetimes.com/document.asp?doc_id=1281322>

[5] "Así es la Kinect por dentro" [en línea]. Febrero 2011. Periódico digital ABC. [consulta 15-3-2014]. Disponible en web: <http://www.abc.es/20110215/tecnologia/abci-kinect-dentro-201102151229.html>

[6] Lowe, D.G.."Distinctive Image Features from Scale-Invariant Keypoints" [en linea]. Vancouver, B.C., Canadá, Enero 2004 [consulta 12-3-2014]. Disponible en web: <http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf>

[7] "Reconocimiento de patrones" [en línea]. Wikipedia.org. [consulta 12-3-2014]. Disponible en web: <http://es.wikipedia.org/wiki/Reconocimiento_de_patrones>

[8] "Scale-Invariant feature transform" [en línea]. Wikipedia.org. [consulta 21-3-2014]. Disponible en web: <http://en.wikipedia.org/wiki/Scale-invariant_feature_transform>

[9] Herigert M.; Bouchet N.; Pianetti I.. "Reconocimiento de Imágenes mediante Scale Invariant Feature Transformation (SIFT)" [en línea]. Facultad Regional Concepción del Uruguay, Uruguay. 2010. [consulta 26-2-2014]. Disponible en web: <http://www.frsf.utn.edu.ar/cneisi2010/archivos/04-Reconocimiento_de_Imgenes_SIFT.pdf>

[10] Pérula, R.. “Estado del arte e implementación de un clasificador de objetos de uso cotidiano”. Director: Abderrahim, M. Tesis doctoral. Universidad Carlos III de Madrid, Departamento de Ingeniería de Sistemas y Automática, Leganés, Madrid, 2013.

[11] Martinez-Ortiz, C.A... “2D and 3D Shape Descriptors” [en línea]. Tutor: Zunic, J. Teis doctoral. Universidad de Exeter, Departamento de Ciencia Computacional, Reino Unido, 2010. [consulta 26-2-2014]. Disponible en: <https://ore.exeter.ac.uk/repository/bitstream/handle/10036/3026/MartinezOrtizC.pdf?sequence=2>

[12] “SURF” [en línea]. Wikipedia.org. [consulta 1-3-2014]. Disponible en web: <http://en.wikipedia.org/wiki/SURF>

[13] Bay. H; Tuytelaars T.; Van Gool L.. “SURF: Speeded Up Robust Features” [en línea]. Katholieke Universiteit Leuven, Zurich, 2006. [consulta 4-3-2014]. Disponible en web: <http://www.vision.ee.ethz.ch/~surf/eccv06.pdf>

[14] Rojo, J.C. “Los usos de la Kinect que nunca llegaron a imaginarse” [on line]. Marzo 2012. Periódico digital EldiarioMontanes.es. [consulta 3-4-2014]. Disponible en web: <http://www.eldiariomontanes.es/v/20120320/cantabria/universidad-cantabria/usos-kinect-nunca-llegaron-20120320.html>

[15] Alexandre, L.A. “3D Descriptors for Object and Category Recognition: a Comparative Evaluation” [online]. 2012. [consulta 3-4-2014]. Disponible en web: <http://www.di.ubi.pt/~lfbaa/pubs/iros2012.pdf>

[16] García, J. “Reconstruction and Recognition of Confusable Models using Three-Dimensional Perception”. Tesis doctoral. Universidad Carlos III de Madrid, Departamento de Ingeniería de Sistemas y Automatización, Leganés, Madrid, 2013.

[17] “Official webpage project handle.” [página web] [cosulta 14-4-2014]. Disponible en web: <http://www.handle-project.eu/>

[18] Ravì, D. “Kinect: the next generation of motion control”. Pp. 12-21.

[19] Zhang, H. “Using Microsoft Kinect Sensor in Our Research”. [en linea]. Universidad de Tennessee. Departamento de Ingeniería Eléctrica y Ciencia Computacional. Knoxville, Septiembre 2011.

[20] Fischler, M.A.; Bolles, R.C. “Random simple consensus: a paradigm for model fitting with applications to image analysis and automated cartography”, Commun. ACM 24 (1981), 381-395.

[21]Jünemann, M. “Object Detection and Recognition with Microsoft Kinect”. Tesis doctoral. Universidad Pública de Berlín. Departamento de Inteligencia de Sistemas y Robótica. Berlin, 2012.

[22] Cantzler, H.; Fisher, R.B.; Devy, M. “Quality enhancement of reconstructed 3D models using coplanarity and constrainsts”, 2002.

[23] Ying, M.; Förstner, W. “Plane detection in point cloud data”, 2º Conferencia Internacional de Guías para Control de Máquinas, 2010, nº 1, 95-104.

[24] Web de Tangible Media Group: http://tangible.media.mit.edu/vision/ [consulta 3-4-2014]

[25] Video] http://vimeo.com/79179138 demostrativo [consulta 3-4-2014]

[26] Wiki sobre curso de ros [en línea] http://robinlab.uji.es/wiki/index.php/Curso_ros:ejerE [consulta 14-5-2014]

[27] López, L.F. “Tutorial de PostgreSQL” [en línea] http://pgsqltutorial.readthedocs.org/en/9.1.0/part_i/index.html [consulta 9-5-2014]

[28] “Manual PostgreSQL 9.1” [en línea] http://www.postgresql.org/docs/9.1/static/index.html [consulta 8-5-2014]

[29] Wiki handle Uc3m [en línea] http://handle2.uc3m.es/trac/wiki [consulta 14-6-2014]

[30] Wiki ROS [en línea] http://wiki.ros.org/ [consulta 14-6-2014]

[31] Foro de preguntas de ROS [en línea] http://answers.ros.org/questions/ [consulta 13-6-2014]

[32] Saponara, F. “Object Recognition using the Kinect” [en linea] http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2011/rapporter11/saponara_favio_11115.pdf [consulta 1-6-2014]. Tesis doctoral. KTH Computer Science and Communication, Stockholm, Suiza, 2011.

[33] Malmsten, P. “Object discovery with a Microsoft Kinect” [en linea]

http://www.wpi.edu/Pubs/E-project/Available/E-project-121512-232610/unrestricted/MalmstenRAILMQP.pdf [consulta 4-6-2014]. Proyecto de cualificación superior, Worcester Polytechnic Institute, 2012.

[34] Wohlkinger, W.; Vincze, M. “3D Object Classification for Mobile Robots in Home-Enviroments using Web-Data” [en linea] http://robots-at-home.acin.tuwien.ac.at/publications/conferences/RAAD10_ww%20copy.pdf [consulta 29-5-2014].

[35] GitHub: Módulos de percepción [en linea] https://github.com/wg-perception/ [consulta 18-5-2014].

8. PRESUPUESTO

En este apartado se calcula una estimación del presupuesto de este proyecto. Se ha realizado una división por componentes necesarios y recursos humanos.

8.1 Presupuesto de material

En la siguiente tabla se recoge el coste del material utilizado para la realización de este proyecto:

Concepto	Cantidad	Precio
Sensor Kinect	1	90€
PC	1	320€
Total		410€

Tabla 3. Presupuesto material

8.2 Presupuesto de personal

En este apartado se detalla el coste del personal durante todo el proyecto. Se hace tenido en cuenta las horas de investigación y las horas de documentación para la preparación del TFG. La hora se ha calculado con un precio base variable en función de la tarea realizada. Las horas de diseño de algoritmo se han omitido ya que no entraba a competencia de este TFG. Sí se han tenido en cuenta las horas de pruebas así como la de reparación de errores en la instalación y durante la ejecución del programa.

Actividad	Nº horas	Precio/h	Coste
Documentación e investigación	250	6	1500
Pruebas	50	10	500
Reparación de errores	180	12	2160
Total	480		4160

Tabla 4. Presupuesto de personal

8.3 Presupuesto final

A continuación se detalla el resumen de todos los gastos recogidos para este proyecto fin de grado.

Tipo de coste	Precio
Coste material	410€
Coste personal	4160€
SUBTOTAL	4570€
I.V.A. (21%)	959,7€
TOTAL	5529,7€

Tabla 5. Presupuesto final

9. ANEXOS

9.1 Instalación uc3m_handle

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
precise main" > /etc/apt/sources.list.d/ros-latest.list'

sudo add-apt-repository ppa:v-launchpad-jochen-sprickerhof-
de/pcl

wget http://packages.ros.org/ros.key -O - | sudo apt-key add
-

sudo apt-get update

sudo apt-get install ros-fuerte-desktop-full

echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc

source ~/.bashrc

sudo apt-get install python-rosinstall

sudo apt-get install ros-fuerte-object-manipulation

sudo apt-get install libopenni0

sudo apt-get install libglew1.6-dev

sudo apt-get install libgsl0-dev

sudo apt-get install ros-fuerte-openni-camera

sudo apt-get install ros-fuerte-openni-kinect

sudo apt-get install ros-fuerte-openni-launch

sudo apt-get update

sudo apt-get install libpcl-all

echo $ROS_PACKAGE_PATH

cd

svn co
https://handle2.uc3m.es/handle/ros/fuerte/uc3m_handle

cd ~/uc3m_handle

svn checkout --revision 2929
https://handle2.uc3m.es/handle/ros/fuerte/uc3m_handle/

mv ~/uc3m_handle/uc3m_handle/perception_pcl17/
~/uc3m_handle/
```

```
sudo rm -R ~/uc3m_handle/uc3m_handle/

cd ~/uc3m_handle/nestk

svn checkout https://handle2.uc3m.es/nestk/ros-electric/

cd ~/uc3m_handle/nestk

mv ros-electric nestk

echo -e "\nexport
ROS_PACKAGE_PATH=~/uc3m_handle:$ROS_PACKAGE_PATH\n" >>
~/.bashrc

source ~/.bashrc

cd

rosmake uc3m_extrectObjTable --pre-clean

rosmake uc3m_objdetect --pre-clean
```

Tabla 6. Código para instalación del proyecto uc3m_handle

9.2 Añadir PATH

```
echo -e "\nexport
ROS_PACKAGE_PATH=~/ros_workspace:$ROS_PACKAGE_PATH\n" >>
~/.bashrc

source ~/.bashrc

#Información de cual es el actual PATH
echo $ROS_PACKAGE_PATH
```

Tabla 7. Añadir PATH

9.3 Modificar PATH

```
sudo gedit ~/.bashrc
```

Tabla 8. Modificar PATH

9.4 Desinstalar ROS

```
sudo apt-get remove ros-*
```

Tabla 9. Desinstalar ROS

9.5 SET Workspace

```
export ROS_WORKSPACE=~/NOMBRE
```

Tabla 10. SET Workspace

9.6 SQL

```
sudo apt-get --purge remove postgresql\*
cd /var/lib/
sudo rm -R postgresql
sudo apt-get install postgresql

cd
sudo su - postgres

control+D

sudo passwd postgres
#Ponemos una contraseña, por ejemplo willow

sudo su - postgres
psql
CREATE ROLE willow LOGIN CREATEDB CREATEROLE PASSWORD
'willow';

control+D
control+D

#Ojo! dos veces control+D, no es un error

sudo gedit /etc/postgresql/9.1/main/pg_hba.conf

#Añadimos las siguientes líneas:

# willow user can connect locally with password
local   all         willow                            md5

#Y modificamos la del postgres poniendo trust en vez de
peer.

ps auxw | grep postgresql
#Nos devolverá, enre otras cosas, 1-3 numeros de procesos
(de 4 dígitos). Los matamos todos con el #siguiente comando:
sudo kill -HUP xxxx

sudo service postgresql reload

sudo su - postgres

dropdb databaseRL
#La hemos borrado por si existiera de antes la base de datos

createdb databaseRL

pg_restore
'/home/mickyubuntu/uc3m_handle/uc3m_household_objects_databa
se/household_objects_handle.backup' --dbname=databaseRL --
username=postgres --password

#La clave que pide es willow

control+D
```

```
sudo apt-get remove pgadmin3
sudo apt-get install pgadmin3

sudo pgadmin3

#Creamos nuevo servidor con los siguientes datos:
#Nombre:databaseRL
#host:localhost
#puerto:5432
#service:
#Maintenance DB: postgres
#username:willow
#password:willow

#Navegamos por el arbol hasta la base de datos databaseRL.
Entramos en su "schemas/public/Tables/" #y hacemos click
derecho sobre "variable". Pinchamos en "View data" y en
"View all rows". Cambiamos #la dirección del PATH y ponemos
"/home/mickyubuntu/uc3m_handle/uc3m_household_objects_databa
se"

#Guardamos y cerramos todo

#Invocamos con ROS la base de datos
roslaunch                    uc3m_household_objects_database
uc3m_database_server.launch
```

Tabla 11. SQL

9.7 Ejemplo Nuevo Workspace con stack de prueba

```
INGRESAR LINEA A LINEA

#Nuevo workspace
rosws init ~/NOMBRE /opt/ros/fuerte

#Directorio "NOVATO2" será un paquete del workspace
mkdir ~/NOMBRE/NOVATO2
source ~/NOMBRE/setup.bash
rosws set ~/NOMBRE/NOVATO2
source ~/NOMBRE/setup.bash

#Nos movemos a Sandbox
cd ~/NOMBRE/NOVATO2

#Creamos el package con sus dependencias
roscreate-pkg NOVATO2 std_msgs rospy roscpp

#Añadimos el paquete al PATH
echo -e "\nexport
ROS_PACKAGE_PATH=~/NOVATO2:$ROS_PACKAGE_PATH\n" >> ~/.bashrc

source ~/.bashrc

#Confirmar que hay un nuevo directorio en el PATH de ROS
```

```
echo $ROS_PACKAGE_PATH

#Comprombamos que se ha añadido el package buscándolo
rospack find NOVATO2

#roscd nos llevará al workspace al que hemos hecho el set, y
pwd nos devolverá su ubicación.
roscd
pwd

#Compilamos con rosmake el paquete
rosmake NOVATO2
```

Tabla 12. Ejemplo de workspace nuevo con stack de prueba

9.8 Editar qué terminal por defecto usar en Ubuntu

```
sudo update-alternatives --config x-terminal-emulator
```

Tabla 13. Editar qué terminal por defecto usar en Ubuntu

9.9 Lanzar Kinect

```
roslauch openni_launch openni.launch

rosrun rviz rviz

rosrun dynamic_reconfigure reconfigure_gui

#Seleccionar /camera/driver/ y marcar la opción de “enable
depth_registration”

roslaunch uc3m_extractObjTable uc3m_extractObjTable.launch

rosservice call /extract_objects_table

roslaunch uc3m_household_objects_database uc3m_server.launch

rosrun uc3m_demo uc3m_demo


rosservice call /start_demo 1
```

Tabla 14. Lanzar Kinect